UNIVERSIDAD CARLOS III DE MADRID

TRABAJO FIN DE GRADO

EMULACIÓN DE TARJETAS INTELIGENTES DE MONEDERO ELECTRÓNICO MEDIANTE TECNOLOGÍA JAVA CARD

GRADO EN INGENIERÍA DE SISTEMAS DE COMUNICACIONES

Autor: Jaime Martínez Puigvert

Tutor: Raúl Sánchez Reíllo

Leganés, 7 de Octubre de 2014

Agradecimientos

Me gustaría utilizar este espacio para agradecer a mi familia y amigos su apoyo y comprensión, tanto durante este proyecto como a lo largo de toda la carrera; sin esos ratos para desconectar habría sido todo mucho más difícil.

También quiero dar las gracias a Raúl, mi tutor, por su ayuda durante este trabajo y durante las prácticas en empresa, y por su paciencia con mis correos a ráfagas.

Gracias a todos.

Resumen

En este trabajo se describe la implementación de una tarjeta inteligente de monedero electrónico mediante el uso de tecnología Java Card, emulando el funcionamiento de una tarjeta con sistema operativo propietario que sigue la norma CEN WG10.

La implementación adecuada del conjunto de directrices que aparecen en los estándares aplicados a este tipo de tarjetas inteligentes es el pilar fundamental sobre el que se articula este proyecto, de manera que, mediante el uso de estas directrices, sea posible establecer un sistema de ficheros y unos mecanismos de seguridad en la aplicación Java Card desarrollada, que permitan efectuar las operaciones que realiza una tarjeta inteligente de monedero electrónico.

Con objeto de comprobar el correcto funcionamiento de la aplicación se han evaluado las respuestas de la tarjeta ante los diversos comandos que soporta y ante los posibles errores en la introducción de estos comandos.

Las características de la tecnología Java Card permiten la interoperabilidad de la aplicación entre tarjetas de distintos fabricantes, manteniendo las funciones criptográficas y de seguridad de la tarjeta con sistema operativo propietario emulada, que garantizan su utilidad como monedero electrónico.

Finalmente se puede concluir que se ha desarrollado una aplicación que cumple con los objetivos planteados, garantizando la seguridad de los datos contenidos en la tarjeta y proporcionando un sistema capaz de gestionar las diferentes operaciones con los ficheros que se definen para la tarjeta con sistema propietario WG10.

Palabras clave: Tarjeta inteligente, monedero electrónico, Java Card, emulación, estándares, sistema de ficheros, sistema de seguridad, comandos, sistema operativo, interoperabilidad, funciones criptográficas.

Abstract

In this project the implementation of an electronic wallet application using Java Card technology is described, emulating the performance of a proprietary operating system smart card that follows the standard CEN WG10.

Proper implementation of the set of rules in the standards applied to this kind of smart cards is the cornerstone in which this project is built, so that by using these guidelines it is possible to establish security mechanisms and file systems in the Java Card application in order to undertake the operations that performs an electronic wallet smart card.

To verify the correct operation of the application, the responses of the card to the various supported commands have been assessed, as well as the possible errors in the introduction of these commands.

The features of Java Card technology allow the application interoperability between different smart cards manufacturers, keeping the cryptographic and security functions of the proprietary operating system emulated card, which makes it useful as an electronic wallet.

Finally, we can conclude that an application that meets the above mentioned objectives has been developed; ensuring the safety of the data contained on the card and providing a system capable of handling different file operations that are defined for the proprietary operating system card WG10.

Keywords: Smart card, electronic wallet, Java Card, emulation, standards, file system, security system, commands, operating system, interoperability, cryptographic functions.

Índice

Índice de Figuras

Índice de Tablas

Listado de Acrónimos

3DES	Tripple Data Encryption Standard
AC1	Condiciones de acceso en actualización
AC2	Condiciones de acceso en lectura
AES	Advanced Encryption Standard
APDU	Application Protocol Data Unit
API	Application Program Interface
ASCII	American Standard Code for Information Interchange
CA	Clave Administrativa
CAP	Converted Applet
CBC	Cipher Block Chaining
CEN	Comité Europeo de Normalización
CPU	Central Process Unit
DES	Data Encryption Standard
DF	Fichero Dedicado
EEPROM	Electrically Erasable Programmable Read Only Memory
EF	Fichero Elemental
EPROM	Erasable Programmable Read Only Memory
GP	Global Platform
I/O	Entrada/Salida

ID	Identificador
ISO	International Standarization Organization
JCRE	Java Card Runtime Enviroment
JCVM	Java Card Virtual Machine
JVM	Java Virtual Machine
MAC	Message Authentication Code
MF	Fichero Maestro
NT	Número de Transacción
PC	Personal Computer
RAM	Random Access Memory
RN	Random Number
ROM	Read Only Memory
SC	Smart Card
SFI	Short File Identifier
SIM	Subscriber Identity Module
SK	Clave de Sesión
SKT	Clave de Sesión Temporal
SM	Mensajes securizados
SOTI	Sistema Operativo de la Tarjeta Inteligente
SW	Status Word
ULE	Unidad de Lectura y Escritura

1 Introducción

1.1 Motivación y objetivos

Una tarjeta inteligente de monedero electrónico es una tarjeta chip con capacidad de almacenar información y de realizar comunicaciones seguras, de manera que se pueda utilizar como medio de pago. La idea original ha sido utilizar este tipo de tarjetas para pagar operaciones con importes bajos, lo que permite que el pago sea más rápido y seguro que en efectivo, evitando la congestión en los puntos de pago, eliminando los problemas de cambio de moneda y aportando mayor seguridad y control en este tipo de transacciones.

Para regular las tarjetas inteligentes con este propósito se ha desarrollado la norma CEN WG10, con las características necesarias para desarrollar un sistema de pago de alta seguridad con bajo coste de operación. Las tarjetas WG10 pueden usarse además en aplicaciones como identificación universitaria, tarjeta de transportes o almacenaje de información segura.

Sin embargo, la programación de tarjetas inteligentes ha sido siempre un proceso complejo, debido a que la mayoría de los entornos de desarrollo utilizan lenguaje ensamblador, y a que consiste en sistemas propietarios, desarrollados por las diferentes empresas del sector. El acceso al desarrollo de estas aplicaciones por parte de terceras partes es muy complicado.

Este problema se soluciona con la aparición de la tecnología Java Card, que permite que las tarjetas inteligentes ejecuten aplicaciones desarrolladas en Java, empleando un subconjunto de este lenguaje. Esto hace que los desarrolladores tengan un acceso más fácil a la tecnología de las tarjetas inteligentes, produciéndose una mayor apertura de las barreras de entrada existentes en este mercado y aumentando el interés en el sector.

La tecnología Java Card se ajusta a los niveles de seguridad exigidos en las tarjetas inteligentes, permite la coexistencia de varias aplicaciones con distintas funcionalidades en la misma tarjeta y posibilita el desarrollo de estas aplicaciones con independencia del fabricante de la tarjeta.

Por tanto, la motivación de este trabajo es el aumento de la interoperabilidad y la flexibilidad de las tarjetas inteligentes de monedero electrónico mediante la emulación de una tarjeta que se ajusta a la norma CEN WG10 usando tecnología Java Card, de manera que, teniendo todas las características en cuanto a funcionamiento y seguridad de la tarjeta WG10, la aplicación Java Card pueda ser instalada a gran escala y en tarjetas de cualquier fabricante e incluso pueda ser instalada junto a otras aplicaciones en la misma tarjeta.

Conforme a esta motivación se establecen los objetivos del trabajo, que asentarán las pautas para el desarrollo del mismo.

El principal objetivo de este Trabajo Fin de Grado, es el desarrollo de una aplicación sobre tecnología Java Card que emule el funcionamiento de una tarjeta inteligente de sistema operativo propietario de tipo WG10, de forma que el funcionamiento de la aplicación y de la tarjeta original sea el mismo de cara al usuario, manteniendo el sistema operativo basado en ficheros y las estrictas condiciones de seguridad de la tarjeta WG10.

Se establecen por tanto tres objetivos cuyo cumplimiento satisfará el propósito de este trabajo:

- Desarrollo en tecnología Java Card de un sistema de ficheros que sea capaz de realizar las mismas operaciones que el sistema operativo de la tarjeta WG10, analizando e interpretando los datos recibidos por la tarjeta de manera que sea posible gestionar el sistema de ficheros, y devolviendo las respuestas y excepciones que devuelve la tarjeta WG10 tras la ejecución de los distintos comandos.
- Desarrollo en tecnología Java Card de un sistema de seguridad que garantice las mismas condiciones que el de la tarjeta WG10, generando las claves necesarias y empleando los mismos algoritmos criptográficos. Además el sistema de seguridad deberá comprobar las condiciones de acceso a los ficheros del sistema y devolver las respuestas y excepciones de la misma forma que lo hace la tarjeta WG10 después de la ejecución de los comandos.
- Por último, será necesaria la integración de los mecanismos de seguridad implementados en el sistema de ficheros desarrollado, de manera que el funcionamiento de la aplicación sea igual al de la tarjeta WG10, realizando también las pruebas y comprobaciones necesarias para asegurar el buen funcionamiento de la aplicación.

1.2 Entorno socio-económico y marco regulador

En la sociedad de la información en la que vivimos hoy en día se está llevando a cabo una revolución en los métodos de pago e identificación, apareciendo nuevas tecnologías en los últimos años que permiten el pago con el teléfono móvil o la identificación a distancia por radio frecuencia.

Lo cierto es que el uso de tarjetas inteligentes o tarjetas chip está ampliamente extendido; como sus principales valedores se puede considerar a las entidades financieras, que en su gran mayoría han sustituido en los últimos años las tarjetas de crédito con banda magnética por tarjetas inteligentes Java Card. Este uso de las tarjetas inteligentes se ha extendido también en otros ámbitos de la sociedad, y son usadas por las administraciones públicas (para el DNI en España por ejemplo), y por las empresas municipales de transporte.

En lo que respecta al marco regulador de las tarjetas inteligentes, cabe destacar las siguientes organizaciones:

- **International Organization for Standarization (ISO):** a través del subcomité SC17 encargado de tecnologías de tarjetas establece diferentes estándares, siendo el más relacionado con este trabajo el ISO/IEC 7816 que incluye el conjunto de normas sobre el funcionamiento de las tarjetas inteligentes.
- **Comité Européen de Normalisation (CEN):** cuyas normas tienen aplicación obligatoria en la Unión Europea.
- **European Telecommunications Standards Institute (ETSI):** instituto europeo con interés en los estándares relativos a las telecomunicaciones.
- **Europay Mastercard y Visa (EMV):** al ser las tres principales entidades en el sector de pagos elaboran también normas para la regulación de las tarjetas financieras.

Además, existen dos organizaciones que elaboran especificaciones sobre las tarjetas inteligentes y la tecnología Java Card:

- **Global Platform:** una asociación sin ánimo de lucro que elabora especificaciones para tarjetas chip, aplicándose en concreto para este proyecto la especificación GP 2.2.
- **Oracle:** publica las especificaciones propias de Java Card, siendo la última versión Java Card Platform Specification 2.2.2.

1.3 Estructura del documento

Este trabajo se estructura en base a siete capítulos que describen las diferentes características y fases del mismo. El primer capítulo es una introducción que presenta al lector los objetivos y las motivaciones de este proyecto así como lo introduce en los aspectos y el entorno de las tarjetas inteligentes y de la tecnología Java Card.

Debido a que la pieza central del trabajo son las tarjetas inteligentes, el capítulo 2 presenta esta tecnología, analizando el estado del arte, y describiendo sus características generales en cuanto a procesamiento, memoria y sistema operativo, lo que permitirá que los siguientes capítulos se afronten con un conocimiento básico del tema. Cabe destacar también la importancia que tienen los estándares en el ámbito de la seguridad de la información, y por tanto esto también se aplica a las tarjetas inteligentes, por eso se incluye en este capítulo una descripción de la normativa aplicable a estos dispositivos.

La tarjeta inteligente de monedero electrónico en cuya emulación consiste este proyecto es la tarjeta WG10, y por esta razón el capítulo 3 se dedica a esta tarjeta, describiendo su sistema de ficheros y su arquitectura de seguridad, características fundamentales a tener en cuenta en este trabajo.

La tarjeta WG10 dispone de un sistema operativo propio, sin embargo, el objetivo de este proyecto es realizar la emulación mediante tecnología Java Card y posibilitar la implementación de las funcionalidades de la tarjeta WG10 en cualquier tarjeta inteligente con sistema operativo Java Card. Con objeto de que el lector comprenda cómo se ha llevado a cabo esta adaptación, el capítulo 4 explica las principales características de la tecnología Java Card.

En el capítulo 5 de este trabajo se detallan las características de la solución adoptada, identificando en primer lugar los componentes necesarios para llevar a cabo dicha solución, analizando los requisitos que debe cumplir y aportando soluciones de diseño a estos requisitos, con el foco en el sistema de ficheros y el sistema de seguridad de la tarjeta. En este capítulo se describen también los comandos implementados en tecnología Java Card y el funcionamiento de la tarjeta ante la recepción de estos comandos.

A consecuencia de su importante papel en este proyecto, se ha dedicado el capítulo 6 a las herramientas de desarrollo utilizadas para llevar a cabo la emulación, haciendo una breve descripción de sus características y su funcionamiento.

Con objeto de documentar las pruebas realizadas con la aplicación desarrollada, se ha incorporado el capítulo 7, que incluye capturas de pantalla de dichas pruebas.

Finalmente en el capítulo 8 se pueden encontrar las conclusiones obtenidas tras la realización de este trabajo y las posibles líneas de futuro en el desarrollo de aplicaciones monedero con tarjetas inteligentes y tecnología Java Card.

Cabe destacar también que en este proyecto se incluye el anexo A, correspondiente a la planificación y el presupuesto del proyecto.

2 La Tecnología de las Tarjetas Inteligentes

2.1 Estado del Arte

Se puede considerar que el origen del uso de las tarjetas como medio de identificación se encuentra en las tarjetas de visita, las cuales almacenan la información mediante la impresión en la superficie de la tarjeta. Hoy en día es sencillo reconocer los inconvenientes de este tipo de almacenamiento, al estar expuesta la información a cualquiera con acceso a la tarjeta, teniendo además grandes limitaciones en cuanto al espacio de almacenamiento y la conservación de los datos impresos (Acedo J. C., 1999).

Desde sus inicios, el uso de las tarjetas como medio de almacenamiento ha evolucionado enormemente, pasando por el desarrollo de tarjetas con almacenamiento óptico, o las conocidas tarjetas de banda magnética, hasta finalmente la aparición de las tarjetas inteligentes, que cuentan con un microprocesador y una memoria con gran capacidad de almacenamiento. Este tipo de tarjetas ha permitido la implementación de fuertes mecanismos de seguridad tanto físicos como lógicos abriéndose camino de esta manera en el sector bancario y de la identificación personal.

A día de hoy, el uso de las tarjetas inteligentes se ha popularizado y extendido a múltiples aplicaciones; prácticamente la totalidad de los cajeros automáticos y de los comercios que aceptan pago electrónico implementan esta tecnología.

Por otro lado se ha desarrollado también un proceso de diversificación de su uso, utilizándose para cualquier tipo de identificación personal, como puede ser una tarjeta de transporte público, un carnet de una universidad o biblioteca, o una tarjeta sanitaria o para el Documento Nacional de Identidad.

Cabe destacar también su importante presencia en las redes de comunicaciones móviles como tarjetas SIM, permitiendo la identificación única de cada abonado en la red e implementando los algoritmos necesarios para dicha identificación.

Con el nacimiento de la tecnología Java Card, en 1996, se ha perseguido el funcionamiento de una misma aplicación en diferentes tarjetas inteligentes con distintos sistemas operativos, de manera que se aumente en gran medida la interoperabilidad, manteniendo a pesar de ello la seguridad de la información almacenada gracias a los algoritmos criptográficos implementados como DES, 3DES, AES o RSA, y el encapsulado de los datos de la aplicación ejecutada.

2.2 Bloques en una Tarjeta Inteligente

Una Tarjeta Inteligente contiene los bloques típicos presentes en todo sistema informático, el circuito integrado que contienen es un micro controlador, es decir, un microprocesador con una memoria y unos periféricos asociados. A continuación se explican con más detalle estos bloques que componen el chip de la tarjeta y, como ejemplo, se muestra la Figura 1:

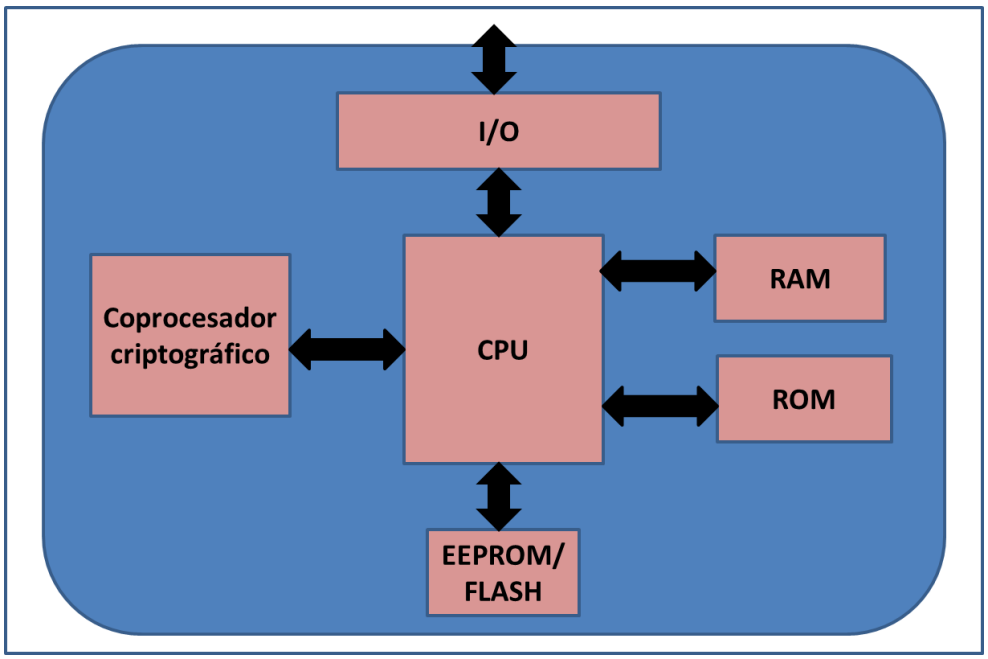

Figura 1 - Bloques en una Tarjeta Inteligente

2.2.1 Unidad Central de Proceso

Este bloque realiza las operaciones a bajo nivel y las interconexiones entre los demás bloques de la tarjeta; además permite la implementación de un Sistema Operativo en la tarjeta; sin embargo, para el usuario este bloque pasa prácticamente desapercibido al no interactuar con él directamente.

Cabe señalar que la evolución en cuanto a capacidad de cálculo de los procesadores presentes en tarjetas inteligentes ha sido bastante reducida en comparación a los procesadores usados en otros dispositivos. Esto es debido a que las aplicaciones para las que son usadas las tarjetas inteligentes no requieren mayores potencias, pudiéndose añadir al circuito integrado en caso de ser necesario coprocesadores matemáticos capaces de implementar los cálculos de algoritmos de cifrado requeridos para una comunicación segura con gran rapidez.

2.2.2 Memorias

Dependiendo de los servicios, existen varios tipos de memoria presentes en las Tarjetas Inteligentes; además se puede decir que este bloque sí que ha tenido una gran evolución a lo largo de los años pasando de las memorias tipo EPROM (las cuales no se pueden borrar de forma eléctrica) a las memorias tipo EEPROM y, actualmente, a las memorias de tipo FLASH.

El abaratamiento del coste de las memorias EEPROM y FLASH, y la reducción de su tamaño permitió que los datos almacenados en las Tarjetas Inteligentes se pudieran sobre-escribir hasta 10000 veces reduciendo también los tiempos de lectura, escritura y borrado. Todos estos avances han permitido que se estén fabricando en la actualidad tarjetas de hasta 128 KB.

Se pueden distinguir tres tipos de memoria presentes en una Tarjeta Inteligente:

- **Memoria RAM o volátil:** Es usada por el microprocesador para realizar los cálculos temporales. En ella guarda las variables necesarias para llevar a cabo dichos cálculos, además; se utiliza para almacenar el buffer de datos I/O. El acceso de esta memoria está restringido tanto al programador de la aplicación de la tarjeta como al usuario final, siendo manejada por el sistema operativo instalado.
- **Memoria ROM:** Es en esta memoria donde está guardado el Sistema Operativo de la tarjeta y las rutinas de arranque. Este Sistema Operativo (SOTI), es desarrollado en un lenguaje de bajo nivel para reducir el tamaño ocupado en memoria. La principal característica de la memoria ROM es que no se puede borrar y al igual que la RAM es transparente tanto para el usuario final como para el programador de la aplicación.
- **Memoria EEPROM (ahora FLASH):** La principal novedad que se introdujo con la aparición de este tipo de memorias es su capacidad de borrarse eléctricamente, siendo además de tipo no-volátil; por tanto será ésta la memoria que verá el usuario final y el programador de la aplicación, ya que almacena los datos generados con la ejecución de la aplicación instalada en la tarjeta. Por esta razón es por lo que su tamaño determina el tamaño real de almacenamiento del que dispone la tarjeta.

  La progresiva sustitución de las memorias EEPROM por memorias FLASH se debe a su bajo coste y a su mayor capacidad de almacenaje; sin embargo presenta los inconvenientes de un acceso a memoria más lento y una menor durabilidad.

2.2.3 Bloque de Entrada y Salida (I/O)

Gracias a este bloque el sistema tiene la posibilidad de comunicarse con el exterior, (por ejemplo mediante un lector de tarjetas). Esta comunicación se llevará a cabo en modo serie, de manera que contiene unos dispositivos encargados de tomar datos en modo serie, realizar un tratamiento de estos datos y posteriormente entregarlos a la CPU.

Para que la comunicación se lleve a cabo de forma correcta y segura, el SOTI debe dedicar específicamente unas rutinas muy robustas que consigan el cumplimiento de los protocolos aplicados a las tarjetas inteligentes.

La característica principal por la que este bloque puede diferir entre unas tarjetas y otras se produce a nivel físico, entre las tarjetas sin contactos (que realizan la comunicación mediante un campo electromagnético cercano) y las tarjetas con contactos que realizan la comunicación mediante la transmisión de señales eléctricas.

2.2.4 Sistema Operativo de la Tarjeta Inteligente (SOTI)

Como ocurre en otros sistemas informáticos, el Sistema Operativo proporciona una interfaz de comando a alto nivel, que facilita el uso de la tarjeta sin permitir en ningún momento la ejecución de instrucciones a bajo nivel que puedan vulnerar la seguridad de la tarjeta, sus principales funciones son (Dreifus H., 1998):

- Controlar el intercambio de datos con el exterior.
- Gestionar los datos y el uso de la memoria.
- Gestionar los mecanismos y servicios de seguridad.

2.3 La Norma ISO 7816

2.3.1 Introducción

La norma ISO 7816 estandariza a nivel internacional las tarjetas electrónicas de identificación con contactos. Este estándar es gestionado de forma conjunta por la Organización Internacional de Normalización (ISO) y la Comisión Electrotécnica Internacional y se encuentra bajo la jurisdicción del comité JTC 1, encargado de normalizar las Tecnologías de la Información. Dentro del JTC1 el subcomité encargado de tecnología de tarjetas es el SC17 (International Organization for Standarization).

Este estándar se subdivide a su vez en 15 partes, sin embargo, en este trabajo no es necesario detallarlas todas, por lo que a continuación se enunciarán las que tienen más relación con el proyecto[1]:

- 7816-1 Características físicas
- 7816-3 Protocolos de interfaz eléctrica y de transmisión
- 7816-4 Organización, la seguridad y los comandos para el intercambio de información
- 7816-8 Comandos para operaciones de seguridad
- 7816-9 Comandos para la gestión de la tarjeta
- 7816-15 Aplicación de información criptográfica

2.3.2 Comunicaciones en las Tarjetas Inteligentes

La unidad de comunicación entre la tarjeta y la unidad de lectura definida por la norma ISO 7816-4 es el APDU. La estructura de un APDU se compone de una cabecera obligatoria de 5 bytes y de un campo de datos con capacidad de hasta 255 bytes (Guthery S. B., 1998). Los campos obligatorios de la cabecera se describen a continuación:

- **CLA (1 byte):** es el primer byte de la cabecera y contiene información del tipo de comando, la seguridad de dicho comando y el canal lógico usado.
- **INS (1 byte):** corresponde al código de instrucción e indica el comando concreto a ejecutar.
- **P1-P2 (2 bytes):** corresponden a los parámetros de ejecución de cada comando en concreto, por lo que se explicará su composición según cada caso.
- **Lc (1 byte):** indica la longitud en bytes que tendrá el campo de datos.

[1] Para una consulta en profundidad de estos estándares ver: International Organization for Standarization, http://www.iso.org/iso/home/store/catalogue_tc/.

-

2.3.3 Seguridad

Una Tarjeta Inteligente, como se ha mencionado anteriormente, contiene mecanismos lógicos de seguridad programados en el microcircuito que evitan interferencias en las comunicaciones, un posible duplicado de la tarjeta, una posible comunicación simulada y provocan la desactivación de la tarjeta en caso necesario.

2.3.3.1 Criptografía

En una comunicación segura es fundamental garantizar la privacidad, la autenticidad y la integridad de los datos transmitidos, y la forma más común para ello es el uso de algoritmos criptográficos, utilizando técnicas de cifrado de datos mediante el uso de claves secretas.

La mayoría de algoritmos de cifrado utiliza claves asimétricas; dos claves diferentes que realiza cada una la operación inversa a la otra. Una de las claves es pública y será difundida entre los receptores y la otra permanece en poder del usuario. El usuario podrá cifrar la información con su clave pública y esa información únicamente podrá ser descifrada por el destinatario que tenga la clave privada correspondiente.

Los algoritmos criptográficos se usan también para proporcionar una huella digital de los datos que se transmiten. Esta huella digital es una cadena de datos de longitud fija obtenida a partir de aplicar una función matemática a los datos; la huella se transmite junto con el mensaje de manera que el receptor puede volver a calcularla y compararla con la recibida para comprobar que los datos no han sido modificados.

2.3.3.2 Mecanismos de Seguridad

Siguiendo la norma ISO/IEC 7816-4 existen los siguientes mecanismos de seguridad utilizados en Tarjetas Inteligentes:

- **Autenticación por contraseña:** Se realiza mediante comandos del tipo VERIFY acompañados de la clave que se desea presentar y se compara con la almacenada. Además se contabiliza el número de intentos de autenticación.
- **Autenticación por clave:** Existen dos tipos: la Autenticación Interna pretende identificar la tarjeta ante el sistema y la Autenticación Externa identifica al sistema frente a la tarjeta. Dependiendo del directorio en el que se ejecute esta autenticación se tiene acceso a unos determinados archivos de la tarjeta. Normalmente la clave utilizada se denomina Clave de Sesión.
- **Mensajes Securizados:** Para la autenticación de los datos se añade información redundante a estos utilizando un algoritmo criptográfico, de manera que se pueda usar también esta información como firma digital. También mediante algoritmos criptográficos y claves la tarjeta es capaz de cifrar la información a enviar.

3 La Tarjeta WG10

La emulación en la que consiste este proyecto tomará como referencia la tarjeta inteligente con monedero electrónico WG10[2], basada en las normas internacionales ISO 7816 y CEN/TC224/WG10 sobre tarjetas inteligentes y monedero electrónico explicadas anteriormente (FNMT Dpto. Tarjetas).

Como ventaja principal, esta tarjeta es capaz de definir diferentes entornos con distintos niveles de seguridad mediante la distribución de la memoria lógica como una estructura en forma de árbol, definiendo dos niveles de jerarquía: un fichero Maestro que hará las funciones de directorio raíz del que dependen ficheros elementales y/o ficheros dedicados, de los cuales pueden depender a su vez otros ficheros elementales.

Al tratarse de una tarjeta diseñada para realizar funciones de monedero electrónico, el sistema operativo propio de la tarjeta incluye mecanismos de autenticación interna y externa, capacidad de intercambio de mensajes securizados y firmas electrónicas. Las operaciones que implementan los mecanismos de seguridad de la tarjeta se basan en la implementación del algoritmo de cifrado Triple DES (3DES).

Con objeto de administrar los ficheros y datos presentes en la tarjeta, el sistema operativo incluye un conjunto de comandos de acuerdo con las normas internacionales sobre tarjetas inteligentes y monedero electrónico que permitirán: realizar operaciones binarias, de registros, de gestión de ficheros, de autenticación y verificación de claves y de inicialización y actualización de parámetros del monedero.

En los siguientes apartados de este capítulo se explicarán con mayor detalle la estructura de ficheros y la arquitectura de seguridad de la tarjeta WG10.

3.1 Ficheros

Como se ha mencionado en la presentación de este capítulo, el sistema operativo de la tarjeta WG10 implementa una estructura jerárquica de ficheros de dos niveles compuesta por los siguientes tipos de ficheros:

- **Fichero Maestro (MF):** representa el directorio raíz, solo existe un MF en la tarjeta.
- **Fichero Dedicado (DF):** puede haber uno o varios DF en la tarjeta; contienen datos relativos a una aplicación como puede ser el monedero; únicamente podrán contener Ficheros Elementales (EF).
- **Fichero Elemental (EF):** los ficheros de este tipo pueden situarse debajo del Fichero Maestro o de Ficheros Dedicados, y contendrán los datos de la aplicación.

[2] En el manual de la tarjeta WG10 se describen con mayor detalle las características expuestas en este capítulo: Dpto. Tarjetas, FNMT *Manual de uso: Tarjeta FNMT WG10.* s.l. : Fábrica Nacional de Moneda y Timbre. Versión 2.0.

3.1.1 Direccionamiento de Ficheros

El sistema operativo de la tarjeta, para poder seleccionar y acceder a los diferentes ficheros, contempla tres formas de direccionamiento:

- **Direccionamiento a través de un identificador:** Únicamente podrán ser seleccionados los ficheros del tipo EF a través de un identificador de dos bytes. Dos ficheros EF en el mismo directorio deberá tener identificadores diferentes.
- **Direccionamiento de un EF a través de un identificador corto SFI:** Un EF puede ser seleccionado mediante el uso de un identificador corto SFI, según especifica la norma ISO/IEC 7816-4, que en el caso de la tarjeta WG10 consistirá en los 5 bits menos significativos del identificador de 2 bytes.
- **Direccionamiento de un DF por nombre:** Un DF o MF puede ser direccionado usando una cadena entre 1 y 16 bytes que servirá como identificador de la aplicación.

3.1.2 Tipos de Ficheros

En este apartado se tratarán los tipos de ficheros elementales que implementa el sistema operativo de la tarjeta WG10 y sus características principales. Será en estos ficheros donde se podrá guardar la información que se genere en la aplicación.

- **Ficheros Transparentes:** En este tipo de ficheros se almacena información no estructurada. Cada fichero se lee y recorre como una secuencia de bytes de datos, los cuales se direccionan mediante un offset según su posición lineal en el fichero.
- **Ficheros con Registros:** Los datos contenidos en este tipo de ficheros son vistos como una secuencia de registros individualmente identificables. En estos ficheros todos los registros tienen el mismo tamaño en bytes y son direccionados en el orden de su creación.

3.2 Arquitectura de Seguridad

La arquitectura de seguridad de la tarjeta WG10 garantiza las operaciones securizadas en la tarjeta, así como la integración de los datos y las operaciones monedero cumpliendo con los estándares de las tarjetas inteligentes y las aplicaciones monedero.

Para la implementación de esta arquitectura de seguridad es fundamental conocer el estado de seguridad de la tarjeta, que representa cómo se encuentra la tarjeta después de ejecutar un comando relacionado con el sistema de seguridad, como puede ser la verificación del código secreto o la prueba de autenticidad de los datos enviados considerando tres posibles estados de seguridad:

- Estado global del código secreto.
- Estado local del código secreto.
- Estado de la clave de sesión.

Para modificar los estados de seguridad en la tarjeta es necesario cumplir ciertas condiciones fijadas por los atributos de seguridad.

3.2.1 Condiciones de acceso a los ficheros

Con objeto de que la tarjeta permita operaciones en los ficheros que contiene, se deben cumplir las condiciones de acceso a cada fichero fijadas por los atributos de seguridad. Estos atributos se definen con la creación de cada fichero y dependen de la categoría y función del fichero. Además, cada condición de acceso a un fichero está definida para un grupo de comandos.

3.2.1.1 Condiciones de acceso AC1 y AC2

Para este caso, siendo AC1 la condición de actualización y AC2 la condición de lectura, el sistema operativo considera cuatro condiciones de acceso diferentes:

- **Libre:** el fichero puede actualizarse o leerse libremente sin necesidad de llevar a cabo ningún procedimiento de seguridad.
- **Código secreto:** para este caso el fichero solo podrá actualizarse o leerse si se ha presentado el código secreto apropiado.
- **Clave secreta:** el fichero en este caso podrá actualizarse usando características de mensajes securizados para verificar la firma.
- **Cerrado:** el fichero en este caso no podrá actualizarse ni leerse.

3.2.1.2 Condiciones de acceso a los ficheros dedicados

Las condiciones de acceso a un DF protegen la creación de ficheros dentro del mismo y el cambio de estas mismas condiciones de acceso. En este caso únicamente se valoran las condiciones en actualización (AC1), ya que en un DF no hay datos accesibles en modo lectura.

3.2.2 Mecanismos de Autentificación

3.2.2.1 Autentificación a través de una contraseña

Este mecanismo de autentificación pretende comprobar el conocimiento de un código secreto, comparando los datos recibidos por el terminal con el código secreto almacenado en la tarjeta. Cabe señalar que únicamente puede haber un código secreto por cada fichero dedicado.

Existe una protección contra la investigación exhaustiva de este código secreto que consiste en el guardado del número de intentos fallidos y el establecimiento de un límite en el número de estos intentos fallidos hasta rechazar el comando.

El código secreto se bloqueará cuando se alcanza el máximo de presentaciones erróneas y únicamente se podrá desbloquear de forma segura utilizando características de mensajes securizados y datos cifrados.

3.2.2.2 Autentificación de un ente por una clave

Este mecanismo de seguridad comprueba el conocimiento de una clave compartida por la tarjeta y el terminal externo con el que se realiza la comunicación. La clave de sesión es una clave secreta pseudoaleatoria almacenada en memoria volátil y recalculada en cada transacción.

Esta clave es también utilizada para cifrar y descifrar datos además de para que el terminal pueda enviar una firma que sea verificada por la tarjeta.

El comando que se usará en este proyecto para el cálculo de una clave de sesión será *"Internal Authenticate*".

3.2.3 Criptografía

3.2.3.1 Triple DES

Como se puede observar en la siguiente figura (Figura 2), la tarjeta WG10 maneja claves de longitud doble llevando a cabo un triple cifrado/descifrado con el algoritmo DES. El algoritmo que compone esta operación se denomina Triple DES.

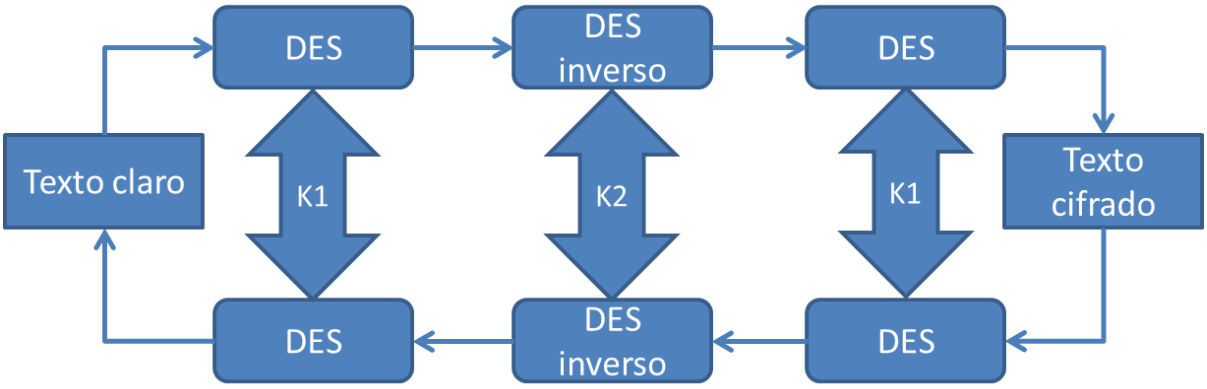

Figura 2 – Algoritmo Triple DES

3.2.3.2 Cifrado/Descifrado de datos confidenciales

Con objeto de proteger los datos confidenciales frente a posibles incursiones externas en la comunicación, se lleva a cabo un cifrado de los datos por parte del terminal usando el triple DES en modo CBC para un posterior descifrado de dichos datos por parte de la tarjeta.

El cifrado en modo CBC consiste en la división del mensaje en bloques y la realización de la operación lógica XOR entre cada bloque y su anterior para posteriormente realizar el cifrado del resultado de la operación. Para la operación XOR con el primer bloque se utilizará un bloque de inicialización todo ceros (Rankl W., 2006).

Como es posible que los datos enviados no sean múltiplo de 8 bytes (número de bytes que componen cada bloque), se completarán los bytes restantes mediante el método *zero-padding*[3].

A continuación, en la Figura 3, se muestra un esquema de la operación:

[3] El método zero-padding utilizado consiste en rellenar con ceros los bits menos significativos del último bloque de 8 bytes, este padding se corresponde con el Método 1 descrito en ISO/IEC 9797.

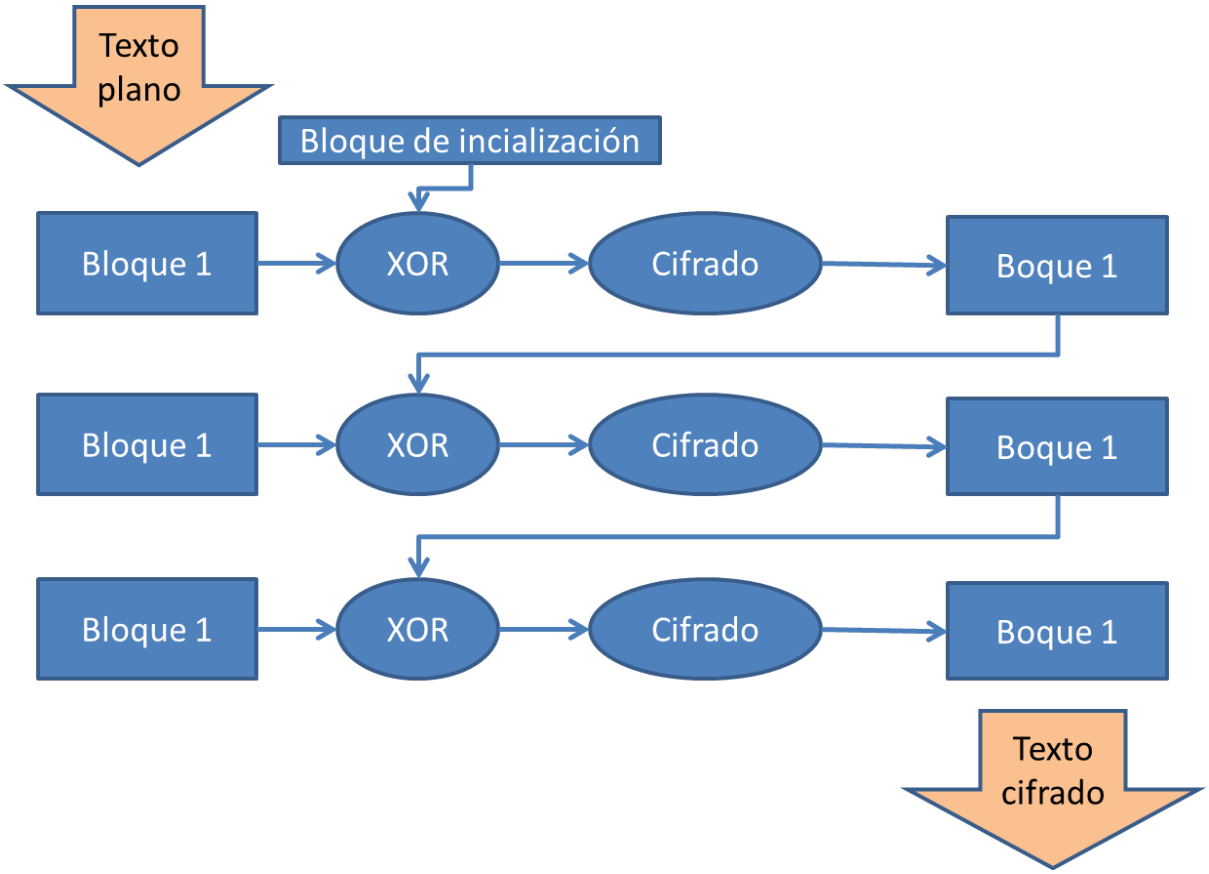

Figura 3 – Operación de cifrado/descifrado

3.2.3.3 Cálculo de claves de sesión

Por razones de confidencialidad, no se puede explicar en este proyecto el método exacto para el cálculo de la clave de sesión, que se realiza de forma completamente segura aplicando el algoritmo Triple DES.

Para el cálculo de la clave de sesión la tarjeta hará uso de la Clave Administrativa (ya almacenada en la memoria de la tarjeta), y del número de transacción (NT) que aumentará cada vez que se calcule una nueva clave de sesión.

3.2.3.4 Cálculo de firmas

Para el cálculo de la firma se utiliza el código de autentificación de mensajes MAC, usando la clave de sesión calculada y verificando de esta manera la veracidad de los interlocutores en la comunicación.

Al ser necesario tener un número de bytes múltiplo de 64, los valores restantes se completan con ceros hasta alcanzar un valor que cumpla el requisito[4]. Entonces se aplica un CBC utilizando DES simple y la primera mitad de la clave sucesivamente hasta el último bloque que se descifrará con la segunda mitad de la clave de sesión mediante un DES inverso y se cifrará el resultado con un DES simple usando la primera mitad de la clave.

El resultado de este proceso será la firma deseada. En el siguiente esquema (Figura 4) se puede analizar con detalle el proceso del cálculo de la firma:

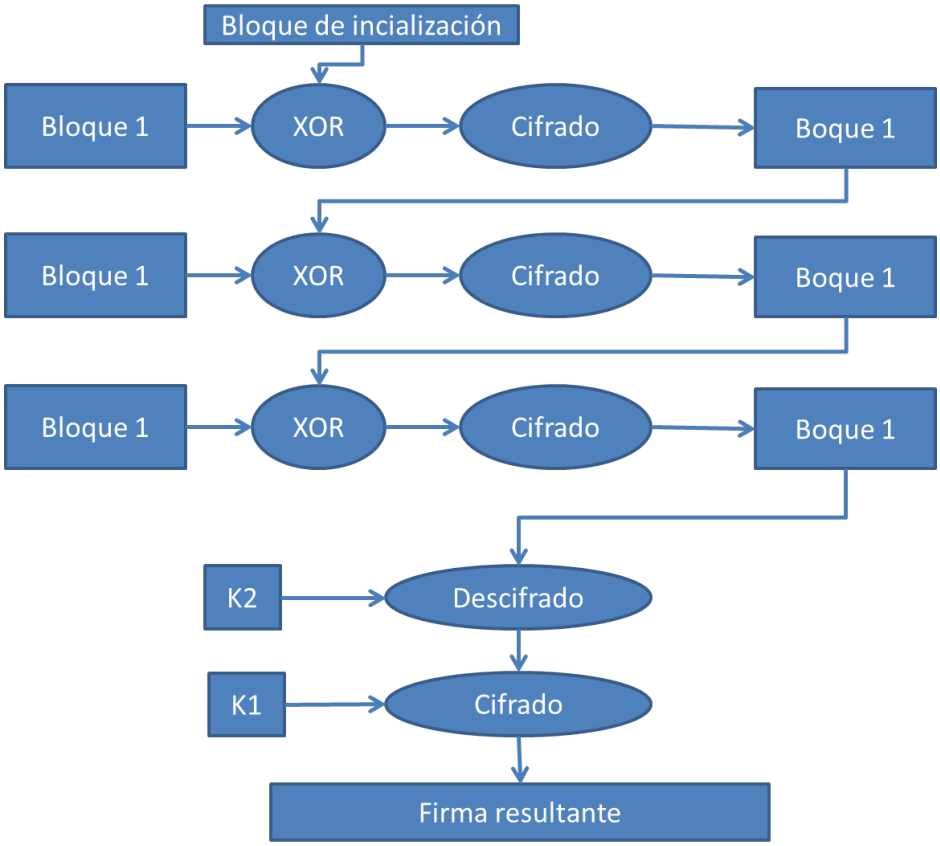

Figura 4 – Cálculo de firmas

[4] Método de zero-padding.

4 Tecnología Java Card

La tecnología Java Card permite ejecutar en tarjetas inteligentes aplicaciones escritas en lenguaje de programación Java. Puesto que la aplicación objetivo de este proyecto será programada en este subconjunto de Java, en este capítulo se hace una introducción a las principales características de esta tecnología (Oracle).

4.1 Arquitectura

Debido a las limitaciones que presentan las tarjetas inteligentes en cuanto a capacidad de memoria y de procesamiento por su pequeño tamaño, el principal reto al que se enfrenta la tecnología Java Card es el introducir el software necesario en la tarjeta dejando siempre espacio suficiente para las aplicaciones.

El problema de espacio en las tarjetas se solventa implementando únicamente un subconjunto de las características del lenguaje Java convencional y aplicando un modelo de división para implementar la JVM (Java Virtual Machine).

Este modelo de división consiste en separar los procesos que correrán *on-card* y *off-card*, es decir, en tiempo de ejecución o no; por ejemplo, tareas que no corren en tiempo de ejecución con tecnología Java Card son: la carga de clases, la verificación de la codificación o la optimización del código.

Además de las características antes mencionadas, la tecnología Java Card define un entorno de ejecución que soporta la configuración de la memoria, las comunicaciones, la seguridad y el modelo de ejecución de aplicaciones de las tarjetas inteligentes conforme al estándar ISO 7816. El entorno de ejecución de Java Card tiene como principal característica la separación entre las aplicaciones y el sistema operativo de manera que define una interfaz a alto nivel a través de la cual las aplicaciones pueden acceder a los servicios del sistema operativo sin gran complejidad (Chen, 2000).

Por tanto la tecnología Java Card se compone de las siguientes tres características principales:

- **Java Card Virtual Machine (JCVM):** compuesta por un subconjunto del lenguaje de programación Java y de la Java Virtual Machine (JVM) adecuado para tarjetas inteligentes.
- **Java Card Runtime Enviroment (JCRE):** describe el comportamiento del entorno de ejecución, incluyendo la gestión de memoria y la gestión de las aplicaciones.
- **Java Card API[5]:** se compone de un conjunto de clases y paquetes que extienden de Java y permiten la programación de aplicaciones en tarjetas inteligentes.

[5] Las librerías de Java Card se pueden encontrar en las referencias (Oracle) y (Chen, 2000).

4.2 Subconjunto del lenguaje Java Card

Ya se han explicado en el apartado anterior las dificultades que conlleva el tan limitado espacio en memoria que tienen las tarjetas inteligentes, y por esa razón, Java Card únicamente soporta un subconjunto seleccionado del lenguaje Java manteniendo la característica de orientación a objetos propia de este lenguaje.

Algunas de las características que no soporta Java Card son: los tipos primitivos de gran tamaño como *long, double* o *float,* los *arrays* de varias dimensiones, la carga dinámica de clases, los hilos o *threads*, o los tipos *char* y *string*[6].

4.3 JCVM

La principal diferencia de la Java Card Virtual Machine con la implementada en Java es que en el caso de Java Card se compone de dos partes separadas como se muestra en la siguiente figura (Figura 5):

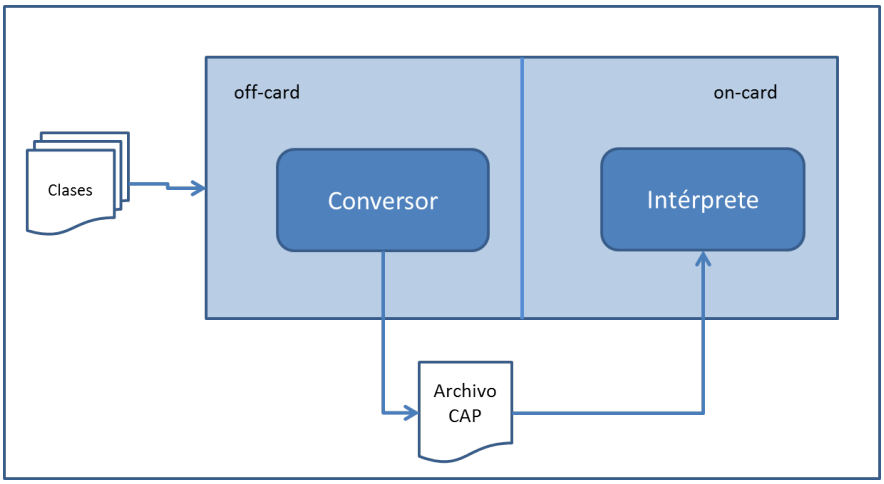

Figura 5 – Java Card Virtual Machine *Fuente:* (Chen, 2000)

Como se puede observar, el conversor es la parte *off-card* de la JCVM. Esto quiere decir que se ejecuta fuera de la tarjeta, en una unidad externa como puede ser un PC. Este conversor carga las clases de Java y las procesa para devolver un archivo CAP (archivo que identifica el *applet* convertido) y un archivo de exportación que contiene la API pública del paquete convertido. El intérprete a su vez es capaz de ejecutar estos archivos CAP generados por el conversor en la tarjeta.

[6] Para consultar todas las características que soporta el subconjunto del lenguaje Java Card ver: Chen, (2000: pp.217-223).

Un archivo CAP contiene una representación binaria ejecutable de las clases en un paquete Java. El formato de los archivos CAP está optimizado para su ejecución en las tarjetas inteligentes usando estructuras compactas de datos.

Los archivos de exportación no son cargados en las tarjetas ni son utilizados por el intérprete, sino por el conversor, para verificar su trabajo. Estos archivos contienen información de la API pública del paquete de un conjunto de clases definiendo el acceso a los métodos y las clases de ese paquete.

El intérprete tiene la característica de ser *on-card*, es decir, que corre en la tarjeta, y es el encargado de ejecutar las instrucciones y las aplicaciones en la tarjeta; también se encarga de la creación de objetos y la gestión de la memoria de las aplicaciones, así como de mantener la seguridad en la tarjeta durante estas tareas.

4.4 JCRE

El Java Card Runtime Enviroment (JCRE) consiste en el conjunto de componentes de un sistema Java Card que se ejecutan dentro de una tarjeta inteligente. Es responsable de la ejecución de la aplicación, de las comunicaciones, de la gestión de los recursos y de la seguridad en las *applets* que tenga instalados la tarjeta. Se puede decir que funciona como el sistema operativo de la tarjeta inteligente. En la siguiente figura (Figura 6) se puede observar una representación de los bloques que componen el JCRE.

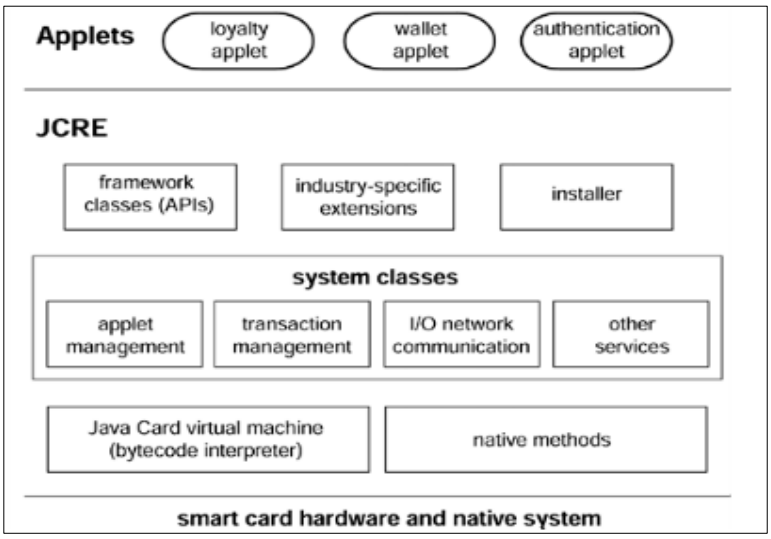

Figura 6 – Java Card Runtime Enviroment. *Fuente:* (Chen, 2000)

El JCRE es capaz de separar las *applets* desarrolladas en Java Card de las propias tecnologías propietarias de los fabricantes de las tarjetas, y provee a las aplicaciones de un entorno normalizado donde ejecutarse.

El intérprete de la JCVM se encuentra al final del JCRE junto con los métodos nativos de la tarjeta. Estos métodos nativos dan soporte tanto a la JCVM como a las clases de bajo nivel del JCRE, manejando la comunicación a bajo nivel en la tarjeta.

El JCRE contiene también las APIs necesarias para la ejecución de las *applets* así como el instalador de nuevas *applets*.

5 Solución Adoptada

5.1 Componentes Necesarios

En este apartado se detallan los componentes necesarios para la implementación de la solución, de manera que la tarjeta inteligente con tecnología Java Card emule el funcionamiento de la tarjeta WG10 y se pueda comprobar y poner en práctica dicho funcionamiento:

5.1.1 Tarjeta Inteligente

La tarjeta inteligente utilizada para el proyecto deberá ser compatible con la tecnología Java Card, preferiblemente con la versión Java Card 3.0.4, al ser la versión más actualizada de la especificación[7].

Por otro lado, será necesario que la tarjeta inteligente utilizada cumpla con las especificaciones de Global Platform, en concreto de la versión GP 2.2[8].

Global Platform es una asociación sin ánimo de lucro que publica especificaciones de seguridad sobre dispositivos chip como pueden ser las tarjetas inteligentes. La versión 2.2 es la versión más moderna de estas especificaciones incluyendo una API más completa y con mejoras en la seguridad de la tarjeta (Global Platform).

También es importante la capacidad de memoria con la que cuente la tarjeta, ya que es esta capacidad la que limita la cantidad de información que se puede almacenar; en este caso, una tarjeta de 32 KB será suficiente para implementar una aplicación monedero como de la que trata el proyecto.

5.1.2 Applet

El *applet* instalado en la tarjeta es el elemento central de este trabajo ya que, gracias al código Java Card en él implementado, es posible emular el comportamiento de la tarjeta inteligente de monedero electrónico WG10.

[7] Esta versión se puede encontrar en: Oracle. http://www.oracle.com/technetwork/java/javacard/specs-jsp-136430.html

[8] Esta especificación para tarjetas inteligentes se puede consultar en: Global Platform, http://globalplatform.org/specificationscard.asp.

Este *applet* está codificado en lenguaje Java Card para poder interpretar los comandos enviados desde una Unidad de Lectura y Escritura (ULE) y realizar las funciones que proporciona la tarjeta inteligente con sistema operativo propio WG10.

El *applet* interpreta los comandos introducidos en código hexadecimal y ejecuta los métodos adecuados para emular el comportamiento de la WG10, como puede ser crear un fichero nuevo en memoria, o la autenticación del usuario para posteriormente poder realizar cambios en un fichero o directorio.

5.1.3 Entrenador de Tarjetas

Se trata de una aplicación instalable en cualquier PC con sistema operativo Windows capaz de enviar y recibir información de una tarjeta inteligente conectada al PC a través de un lector de tarjetas.

Esta aplicación proporciona una interfaz sencilla de manejar desde la que se pueden enviar comandos a la tarjeta conectada siguiendo los estándares sobre tarjetas inteligentes; además se pueden guardar los comandos utilizados durante una sesión en un fichero o cargar ficheros de comandos para ejecutarlos sobre la tarjeta directamente.

El entrenador de tarjetas permite también la traducción de código ASCII a Hexadecimal y viceversa, lo que resulta de gran ayuda al grabar información en la tarjeta, pues toda información almacenada en la tarjeta se guardará en formato Hexadecimal.

A continuación se muestra la interfaz del entrenador de tarjetas:

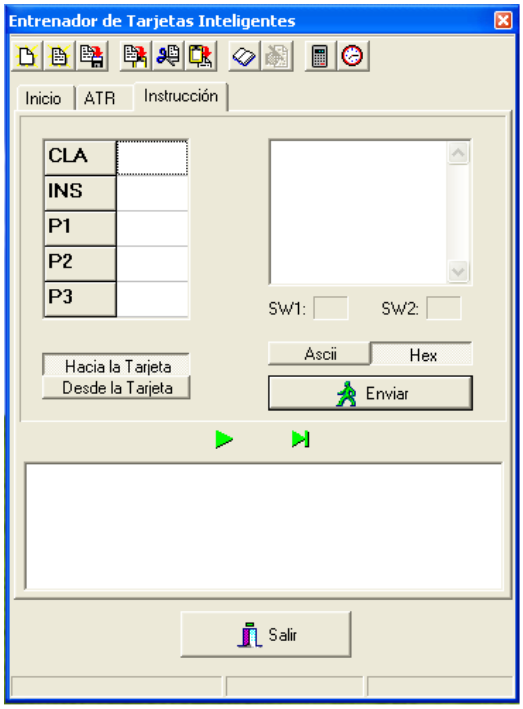

Figura 7 – Entrenador de tarjetas

En la Figura 7 se puede observar cómo se pueden introducir en el entrenador de tarjetas los bytes correspondientes a la cabecera de comandos a la izquierda y los datos, en caso de ser necesarios en el espacio de la derecha. Mediante el entrenador de tarjetas es posible enviar y recibir datos, mostrándose la respuesta de la tarjeta en el espacio en blanco de la parte inferior de la imagen. Se pueden también generar ficheros con una secuencia de comandos que se ejecutarán sucesivamente en la tarjeta.

Además para la realización de este proyecto se ha utilizado otra herramienta para el cálculo de los algoritmos criptográficos necesarios, el cálculo de las claves y el proceso de firmado, cifrado y descifrado. El uso de esta otra aplicación[9] ha permitido ganar tiempo en la realización de los citados cálculos.

5.1.4 Lector de Tarjetas

El lector de tarjetas o Unidad de Lectura y Escritura (ULE) es el dispositivo que conecta físicamente el PC con la tarjeta inteligente (dispositivo PC/SC), y es controlado por la aplicación instalada en el PC para comunicar con la tarjeta inteligente (el Entrenador de Tarjetas).

Para este trabajo se ha utilizado el lector de tarjetas SCM Microsystems Inc. SDI010, que tiene capacidad para tarjetas con contactos y sin contactos si bien para este proyecto se ha usado una tarjeta con contactos para la emulación de la tarjeta WG10 (también con contactos).

5.2 Estructura de Ficheros

La estructura que tiene el sistema de archivos que implementa la aplicación, es uno de los puntos clave para la correcta emulación de la tarjeta WG10, dado que las demás características que tiene la aplicación se construyen basándose en cómo se ha diseñado este sistema de archivos.

El sistema de archivos de la aplicación debe emular completamente el sistema de la tarjeta WG10. Esto no quiere decir que deba copiarse el sistema de archivos de la WG10, sino que, desde el punto de vista de un usuario de la tarjeta Java Card, el funcionamiento de las dos tarjetas tiene que ser el mismo. Debe tener las mismas funcionalidades que la tarjeta con sistema operativo propio, pero realizar estas funcionalidades gracias a la tecnología Java Card.

[9] Las dos aplicaciones descritas en este punto han sido proporcionadas por el tutor de este TFG: Raúl Sánchez Reillo.

Prácticamente la totalidad de los comandos[10], utilizan el sistema de ficheros, bien sea para crear un fichero, bien para actualizarlo o bien para obtener información de él. El uso del sistema de ficheros, del direccionamiento, de la búsqueda, y de la escritura/lectura en memoria, es casi constante durante el uso de la tarjeta.

5.2.1 Requisitos

El análisis de los requisitos es fundamental para conocer perfectamente las funcionalidades que debe cumplir la aplicación, y de esta manera enfrentar el problema con un conjunto de puntos clave que se deben cumplir para la consecución del objetivo de, en este caso, la creación del sistema de ficheros que utiliza la aplicación.

En este apartado se explican los requisitos de diseño que se han tomado en cuenta para la implementación del sistema de ficheros que tiene la aplicación Java Card desarrollada.

- En primer lugar, el sistema de archivos debe ser capaz de almacenar dos tipos de información: los datos que contengan los propios ficheros, y la información referente a las características de cada fichero.

  La información que contendrán los datos de cada fichero es trasparente para el sistema y la maneja como un conjunto de bytes sin analizarla; pero la información referente a las características de los ficheros almacenados en el sistema sí es analizada durante los diferentes procesos que lleva a cabo la aplicación.

- Por otro lado el sistema tiene que poder guardar los ficheros y la información contenida en ellos durante la sesión iniciada por el usuario y cuando se desconecte la tarjeta, esto es, los ficheros y los datos deberán permanecer guardados en la tarjeta íntegramente sin ninguna alteración ante cualquier conexión o desconexión de la tarjeta.

- Lo mismo que se ha referido en el punto anterior se aplica también a la información que contiene las características de los ficheros almacenados (por ejemplo el identificador de fichero, o las condiciones de acceso en lectura o en actualización). Esta información debe almacenarse también soportando los efectos de las posibles desconexiones y conexiones de la tarjeta.

- El sistema de ficheros, también debe ser capaz de llevar a cabo el direccionamiento de los ficheros correctamente, es decir, debe poder identificar claramente de cada fichero, sus propiedades y características, y los datos almacenados en el fichero; y poder situar correctamente todos los datos asociados al fichero en la memoria de la tarjeta para ser accedidos ya sea por necesidades del sistema o por requerimiento del usuario.

- En relación con el punto anterior, el sistema será capaz también de encontrar un fichero en la memoria de la tarjeta a partir de un identificador dado. La diferencia con el punto anterior es que en este caso se trata de localizar el propio fichero en la memoria y no los datos asociados al fichero.

[10] En el apartado 5.4 se hace una descripción detallada de los comandos implementados.

- Será posible analizar el identificador del fichero y compararlo con los identificadores de los ficheros ya existentes en la memoria, devolviendo una excepción en caso de coincidencia.
- El sistema controlará también las condiciones de acceso y el propio acceso a los ficheros mediante el análisis de las propiedades de cada fichero.
- Por último, el sistema deberá implementar todas estas funciones de una forma robusta, en ausencia de fallos de memoria y con todas las posibles excepciones controladas. El uso de la memoria de la aplicación deberá estar completamente definido para su correcto funcionamiento.

5.2.2 Diseño

Para el diseño del sistema de archivos de la aplicación se han tomado en cuenta los requisitos identificados previamente[11], de manera que el sistema realice todas las funciones que tiene la tarjeta WG10, cumpliendo con los requisitos mencionados para asegurar la robustez y fiabilidad del sistema.

Se ha decidido implementar un sistema basado en *arrays*, esto es, estructuras lineales de datos que almacenan los bytes y hacen las funciones de la memoria de la aplicación.

Estos *arrays* desarrollados en lenguaje Java son de tipo *byte* para almacenar sin fallos de compatibilidad los datos recibidos por la tarjeta del terminal, aunque en caso de ser necesario, se realizan las oportunas conversiones de los datos a otro tipo como puede ser *short* o *int* para realizar cálculos sobre el sistema de ficheros, almacenando en este caso los datos convertidos en variables específicas.

No se han utilizado otras estructuras de datos como pueden ser *arrays* multidimensionales, listas enlazadas o una estructura en árbol, porque el subconjunto del lenguaje Java que forma Java Card, no permite el uso de estas estructuras de datos debido a las restricciones de espacio en la memoria de la tarjeta y el gran espacio que estas estructuras ocuparían.

Otra posibilidad hubiera sido el desarrollo de una estructura de datos que combinara un *array* de memoria que almacenara los diferentes ficheros, y que estos ficheros fueran en realidad objetos, es decir, instancias de una clase de Java con atributos que guardaran las características de cada archivo, así como los datos que contiene.

Sin embargo una estructura de este tipo, presentaría varios inconvenientes: en primer lugar la creación dinámica de objetos (cada vez que se invocara el comando *Create File* se crearía un objeto), podría presentar problemas y ralentizaría el sistema operativo Java Card. Por otro lado, estos objetos no serían entidades de un espacio en memoria normalizado, ya que los ficheros se pueden crear de distintos tamaños.

Esto provocaría que se tuvieran que inicializar todos los objetos de tipo fichero del mismo tamaño (el tamaño máximo de un fichero para la WG10) y que después se rellenara solo el

[11] Véase apartado 5.2.1.

espacio que se fuera a utilizar del fichero; o que el tamaño de los objetos creados como ficheros fuera variable, lo que provocaría realizar reservas dinámicas de memoria que ralentizarían y podrían provocar excepciones en el sistema operativo de la tarjeta Java Card.

Es importante que la estructura de ficheros sea robusta y que las excepciones (respuestas ante los errores) que se puedan producir, derivadas de su uso, estén controladas. Asimismo es fundamental que la información que se ofrezca al usuario sobre estas excepciones, se proporcione de la misma manera que lo hace la tarjeta WG10. Por todas estas razones finalmente se ha implementado una estructura de ficheros basada en *arrays* de una sola dimensión.

El sistema de ficheros desarrollado consta de tres *arrays* principales que constituyen la base de la estructura del sistema, y varias variables y *arrays* auxiliares utilizados para realizar operaciones con los datos almacenados en los tres *arrays* principales.

Estos tres *arrays* principales son: el *array de direccionamiento*, el *array de memoria* y el *array de selección.*

La estructura de la memoria del sistema está diseñada de forma que hay un *array de memoria* donde están almacenados todos los datos existentes en la aplicación, sin tener a simple vista una lógica de distribución de los bytes definida, si no que están todos los datos de la aplicación unos detrás de otros.

La lógica de cómo está distribuida la memoria viene establecida por el *array de direccionamiento*. Este *array* sí que tiene una disposición lógica determinada de sus datos, y guarda, para cada identificador de fichero, sus características de acceso, tipo de fichero o posición en memoria, de manera que a través del *array de direccionamiento* se pueda acceder a los datos de cada fichero almacenados en el *array de memoria.*

Por último, el *array* de selección *read* tiene el tamaño exacto para almacenar los datos de direccionamiento de un único fichero, y se usa en caso de ser necesario para seleccionar un fichero y almacenar sus datos y no tener así que volver a recorrer el *array de direccionamiento* para buscarlo de nuevo; vale con acceder al *array read*. Se utiliza sobre todo para la selección directa (o explícita) de ficheros.

En los siguientes puntos se explica con más detalle el diseño de cada uno de los *array* que forman la estructura del sistema de ficheros de la aplicación.

5.2.2.1 Array de direccionamiento

El *array de direccionamiento* contiene la información de los ficheros que están guardados en la memoria de la aplicación, de manera que se pueda acceder a las propiedades de cada fichero o, mediante la referencia guardada en este *array*, a los datos del fichero contenidos en el *array de memoria*.

La información referente a cada fichero se almacena linealmente en el *array de direccionamiento* en grupos de 12 bytes que contendrán los datos de las propiedades del fichero al que se refiere cada grupo.

Esta información referente a cada fichero almacenada en el *array de direccionamiento* está compuesta por:

- **Identificador del fichero:** forma los dos primeros bytes de un grupo referente a un fichero. Identifica al fichero de forma única de manera que es utilizado para buscar un fichero en el *array de direccionamiento* y así localizar el resto de datos referentes al fichero identificado.
- **Tipo:** ocupa el tercer byte de un grupo referente a un fichero en el *array de direccionamiento*. Identifica el tipo del fichero almacenado como, por ejemplo, fichero transparente binario o fichero con registros. Es de gran importancia identificar el tipo de fichero de que se trata para efectuar posteriormente operaciones sobre el fichero, cuyo proceso varía según su tipo; además resulta fundamental para realizar las comprobaciones necesarias y determinar los privilegios de acceso al fichero.
- **Opciones:** el byte de opciones almacena diferentes datos dependiendo del tipo de fichero del que se trate, en un fichero con registros almacena la longitud de los registros.
- **Inicio:** los dos bytes de inicio indican la posición donde comienza el fichero dentro del *array de memoria*. Son una referencia fundamental para acceder a los datos almacenados en el fichero. Lo componen dos bytes para poder acceder a suficientes posiciones dentro del *array de memoria*.
- **Tamaño:** también compuesto por dos bytes, indica el número de bytes de datos del fichero almacenados en el *array de memoria*. Junto con la información del inicio del fichero en el *array memoria*, se pueden conocer las posiciones exactas que ocupan los datos del fichero en memoria y, por tanto, el byte en el cual empezará en memoria el siguiente fichero.
- **Actualización:** este byte almacena las condiciones en actualización del fichero al que se refiere. Podrá ser consultado y modificado si se cumplen las condiciones de acceso.
- **Lectura:** este byte almacena las condiciones en lectura del fichero al que se refiere. Podrá ser consultado y modificado si se cumplen las condiciones de acceso.
- **Bytes extra:** en cada módulo que compone el direccionamiento de un fichero en la memoria, se incluyen dos bytes extra para posibles implementaciones de funciones nuevas o para funciones y propiedades especiales que únicamente se aplican a algunos tipos de ficheros. Como ejemplo está el dato del número de registros añadidos en un fichero con registros (*append*), que es necesario para la emulación de la tarjeta WG10, pero no se aplica esta operación a todos los tipos de ficheros.

En la siguiente figura (Figura 8) se muestra la disposición de la información referente a un fichero en el *array de direccionamiento*. Con la creación de un fichero se añaden, al final del *array*, los datos del nuevo fichero creado.

ID1	ID2	Tipo	Opción	Inicio	Inicio	Tamaño	Tamaño	Actuali-zación	Lectura	Extra	Extra

Figura 8 – *Array de direccionamiento*

De esta manera la estructura de los datos contenidos en el *array de direccionamiento* estará completamente definida y será homogénea para todos los ficheros contenidos en la aplicación. Esto es importante, ya que facilita el recorrer este *array* para buscar los ficheros contenidos en la tarjeta y la información asociada a estos ficheros.

5.2.2.2 Array de memoria

El *array de memoria* contendrá los datos de todos los ficheros almacenados en la aplicación, que estarán referenciados por los datos correspondientes a cada fichero en el *array de direccionamiento.*

En este *array de memoria* los datos están almacenados linealmente, sin ningún tipo de cabecera o de indicador que diferencie unos de otros o los pertenecientes a un fichero o a otro, de manera que el acceder a esta memoria directamente no permitiría interpretar los datos en ella contenidos.

Como se ha explicado en el punto anterior, es el *array de direccionamiento* el que permite al sistema interpretar la distribución de los datos dentro del *array de memoria* y, sin una correcta coordinación entre los dos *arrays*, no se podría tener un acceso a los datos de forma fiable y robusta ante posibles fallos.

Esto es debido a que cada fichero contenido en la memoria puede ocupar un espacio en bytes diferente de los demás ficheros, lo que provoca que no exista una estructura homogénea en el *array de memoria* y que, por tanto, no se pueda seguir una lógica sencilla para buscar cada fichero dentro de este *array*, como se muestra en la figura (Figura 9), donde se pueden observar las diferentes dimensiones de cada fichero contenido en memoria.

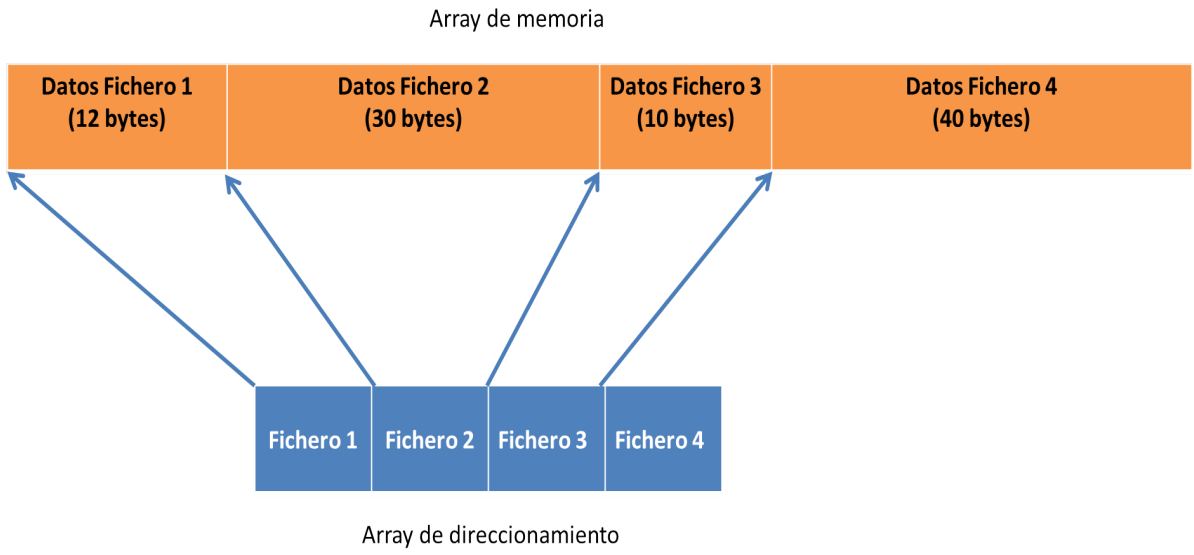

Figura 9 – Sistema de archivos de la aplicación

En la figura se muestra además cómo a través de una memoria estructurada y de menor tamaño como es el *array de direccionamiento*, se referencia a otra memoria con estructura variable y de mayor tamaño como es el *array memoria*, de manera que se pueda mapear el contenido del *array memoria* mediante el uso de la información contenida en el *array de direccionamiento.*

5.2.2.3 Array Read

El *array read* tiene gran relevancia en el funcionamiento de la aplicación, ya que en él se almacena el resultado del comando *Select File*[12] (que selecciona un fichero de la memoria de la tarjeta).

El comando *Select File* tiene como función seleccionar un fichero de la memoria de la tarjeta para después realizar operaciones sobre este fichero mediante el uso de otros comandos. Por esa razón es necesario que exista en la aplicación alguna forma de seleccionar un fichero y mantener esta selección a pesar del final de la ejecución del comando *Select File.*

Por ello, tras la ejecución de *Select File,* el *array read* almacena la información que existe referente a un fichero en el *array de direccionamiento*. Esto permite posteriormente poder localizar los datos del fichero en memoria a través de la información guardada en el *array read*.

El contenido del *array read* se borrará una vez se haya ejecutado el comando siguiente a *Select File*, para que no siga seleccionado el mismo fichero durante la ejecución de los comandos que vengan después.

5.2.2.4 Elementos auxiliares

Como elementos auxiliares del sistema de ficheros de la aplicación, se ha clasificado a un conjunto de variables y *arrays* que se utilizan para llevar a cabo las operaciones que debe realizar el sistema, como puede ser el recorrer los *arrays* que componen la memoria, o almacenar momentáneamente determinados datos de la información de los ficheros.

Estos elementos auxiliares son:

- **Punteros de memoria:** son variables de tipo short que hacen las funciones de punteros para recorrer los *arrays de memoria* y *de direccionamiento*. Estos punteros indican en su valor la posición del *array* donde se encuentran. De esta manera se puede recorrer la memoria de la aplicación y obtener la posición de unos datos y un fichero determinados. Se utilizan también dos punteros para indicar la primera posición vacía en cada uno de los dos *arrays*. Esta posición se incrementa a medida que se almacenan ficheros en la aplicación.
- ***Array* ID:** debido al uso que se le da al identificador de un fichero durante las operaciones de la aplicación, se ha creado un *array* que guardará específicamente este identificador.

[12] Véase apartado 5.4.1.2.

- ***Array response*:** como resultado de ciertos comandos, la tarjeta genera un conjunto de datos que pueden ser obtenidos por el terminal utilizando el comando *Get Response*[13]. El *array response* almacena estos datos para proporcionarlos en caso de que el terminal los solicite. Esta forma de actuar, se debe a que la tarjeta WG10 implementa el protocolo T=0[14]. Según este protocolo, no se permite transmitir y recibir datos durante la ejecución de un mismo comando, por lo que el *array response* almacena temporalmente los datos de respuesta.

5.2.3 Funcionamiento

En este apartado se explica el funcionamiento del sistema de ficheros de la aplicación para tres casos generales en los que la tarjeta almacena información o accede a la información guardada en memoria. Estos pasos son realizados numerosas veces durante la ejecución de la aplicación.

5.2.3.1 Creación de ficheros

La creación de ficheros es la función a partir de la cual se comienza a usar la tarjeta WG10 y, por tanto, la emulación en Java Card, ya que a partir de la creación de un fichero se pueden realizar otras operaciones con la tarjeta.

Al crear un fichero se inicializan todas sus propiedades y se le asigna un identificador único que permitirá volver a seleccionar y utilizar dicho fichero las veces que sea necesario.

En primer lugar, cuando la tarjeta recibe el comando *Create File*[15], hace las comprobaciones pertinentes con las opciones indicadas en el comando que ha recibido, para asegurar que las características que tenga el nuevo fichero creado sean compatibles entre sí.

En segundo lugar, se guarda el identificador de fichero en un *array* auxiliar de dos bytes y se recorre el *array de direccionamiento* analizando todos los identificadores de los ficheros creados anteriormente. En caso de coincidencia, se rechaza el comando, al no poderse crear ficheros con dos identificadores iguales.

Una vez se ha asegurado que el identificador de fichero es único en la tarjeta, se procede a rellenar las posiciones que ocuparán las propiedades del fichero en el *array de direccionamiento*, comenzando por el identificador de fichero.

Para determinar la posición en la que se empezará a escribir en el *array de direccionamiento* se consulta la variable que hace las funciones de puntero indicando la primera posición vacía del *array*.

[13] Véase apartado 5.4.2.3.
[14] Para más información: J. C. Acedo et al. (2000: pp. 76-80).

[15] Véase apartado 5.4.1.1.

Además, se consulta también la variable que hace las funciones de puntero para el *array de memoria* e indica la primera posición vacía de este *array*, de manera que se determine la posición de inicio de los datos del fichero para incluirla en la información que ha de insertarse en el *array de direccionamiento*.

En el comando de creación de ficheros (*Create File*) se indica el tamaño del fichero que se va a crear, de manera que se almacena este valor para introducirlo con las características del fichero en el *array de direccionamiento*. Este valor sirve también para modificar la posición del puntero que indica dónde comenzar a escribir en el *array de memoria*, haciendo espacio para los datos de este nuevo fichero creado.

Finalmente se introducen todas las propiedades del fichero que venían con el comando y las calculadas posteriormente en las 12 primeras posiciones vacías del *array de direccionamiento*[16], y se actualiza la posición del puntero de este *array*.

Nótese que con la creación de un fichero no se alteran para nada los datos almacenados en el *array de memoria* y que únicamente se introduce información en el *array de direccionamiento.*

5.2.3.2 Selección de ficheros

Para la selección de ficheros se hace uso del *array auxiliar read*, que guardará todas las propiedades de un fichero que se almacenan en el *array de direccionamiento*.

En primer lugar se hacen las comprobaciones pertinentes sobre el comando introducido asegurando que los datos son correctos.

En segundo lugar, se almacena el identificador del fichero que se ha de buscar y se recorre el *array de direccionamiento* hasta encontrar una coincidencia. En caso de que no esté almacenado en la tarjeta ningún fichero con ese identificador, se devolverá la excepción correspondiente.

Cuando se encuentra el identificador correspondiente en el *array de direccionamiento*, se copia la información de ese fichero disponible en el *array auxiliar read*, de manera que el fichero queda seleccionado ya y guardadas sus propiedades para cuando se ejecute el siguiente comando.

El proceso de selección implícita se lleva a cabo de la misma manera, salvo por el hecho de que no se almacena la información del fichero de forma permanente en el *array read*, sino que se almacena en un *array* temporal mientras se realizan las operaciones que determinen el comando llamado.

[16] Como se indica en el apartado 5.2.2.1.

5.2.3.3 Lectura y escritura de ficheros

Para la lectura y la escritura de ficheros se sigue un procedimiento muy similar, ya que en los dos casos se accede a los datos del *array de memoria*.

Después de realizar las operaciones de comprobación de los comandos introducidos, se procede a verificar si existe un fichero seleccionado, bien sea mediante selección implícita o bien mediante el comando *Select File*. En caso de no haber ningún fichero seleccionado, se devolverá la excepción correspondiente.

Tras tener un fichero seleccionado, se accede a los datos dentro de las propiedades del fichero que indican las posiciones en memoria que ocupa dicho fichero, de manera que se pueda acceder a ellas mediante la ejecución del comando.

El propio comando indica el offset que se aplicará en el acceso al fichero y, utilizando este offset, o bien se modificarán los datos correspondientes del fichero en memoria, o bien se copiarán estos datos en la respuesta de la tarjeta para que el usuario pueda leerlos.

En el caso de ficheros con registros, se accederá al registro correspondiente indicado en el comando de lectura o escritura y se hará la operación que se indique en el comando. Previamente se comprobará si el registro correspondiente ha sido inicializado mediante el comando *Append Record*[17].

Tanto la función de lectura como la de escritura en ficheros realizan el mismo proceso hasta llegar a un punto en el que, con el *offset* determinado dentro del fichero, para la lectura se devuelven los bytes seleccionados al usuario, y para la escritura en el fichero se sobrescriben los bytes seleccionados por los introducidos por el usuario.

5.3 Seguridad en la Tarjeta

La seguridad en los ficheros es un requisito fundamental para las tarjetas inteligentes de monedero electrónico como la WG10 y, por tanto, lo es también en la aplicación desarrollada para la emulación de esta tarjeta. Se deben garantizar las operaciones de seguridad en la tarjeta, como las transacciones monedero, y también se debe garantizar la integridad de los datos.

Dos puntos clave definen la seguridad en la aplicación: el *estado de seguridad* y los *atributos de seguridad.*

El *estado de seguridad* representa el estado en que se encuentra la tarjeta después de ejecutar un comando de seguridad; por ejemplo, después de realizar el cálculo de la clave de sesión, este estado puede cambiar en caso de que se seleccione otro directorio, con lo que se perdería la clave de sesión calculada.

Por otro lado, los atributos de seguridad fijan las condiciones que deben cumplirse para permitir el acceso y las operaciones en los ficheros de la tarjeta; de esta manera, cada fichero alojado en la tarjeta lleva asociadas unas condiciones de acceso y cada condición está definida para un grupo de comandos (comandos de actualización y de lectura).

[17] Véase apartado 5.4.1.5.

En este apartado se explica qué requisitos se han tenido en cuenta y cómo se ha llevado a cabo la implementación del sistema de seguridad en la aplicación Java Card.

5.3.1 Requisitos

Con objeto de conocer las funcionalidades en materia de seguridad que debe implementar la aplicación, se ha hecho un análisis de los requisitos que debe cumplir el sistema de seguridad de la tarjeta y que se explican a continuación:

- La aplicación debe ser capaz de analizar y diferenciar las diversas condiciones de acceso a los ficheros y asociar correctamente las condiciones a cada fichero correspondiente, relacionando también estas propiedades con el tipo del fichero en cuestión.
- Asimismo será necesario que la aplicación pueda discriminar y aceptar o rechazar el acceso, dependiendo de si se satisfacen o no las condiciones de acceso establecidas. Este proceso es fundamental para mantener la seguridad en los ficheros guardados en la aplicación y las funcionalidades de monedero electrónico de la tarjeta.
- Debe ser posible también modificar las condiciones de acceso a un fichero, siempre dentro de los casos contemplados para la tarjeta WG10. En la Tabla 1 se muestran las reglas para el cambio en las condiciones de acceso de un fichero.

		Nueva condición de acceso		
		Código Secreto	Clave Secreta	Cerrado
Antigua condición de acceso	Libre	OK	OK	OK
	Código Secreto	OK	OK	OK
	Clave Secreta	NO	OK	OK
	Cerrado	NO	NO	OK

Tabla 1 – Condiciones de acceso a ficheros. *Fuente:* (FNMT Dpto. Tarjetas)

- La aplicación también deberá calcular una clave de sesión en caso de que se ejecute el comando *Internal Authenticate*[18]. Esta clave de sesión deberá poder utilizarse después de su cálculo para realizar otras funciones de seguridad en la tarjeta[19].
- Para poder hacer uso de los mensajes securizados, la aplicación debe ser capaz de calcular una firma a partir de unos datos dados, usando una clave de sesión hallada anteriormente[20].

[18] Véase apartado 5.4.2.1.
[19] El proceso de cálculo de la clave de sesión se hará conforme a lo explicado en el apartado 3.2.3.3.
[20] Esto se hará mediante el proceso explicado en el apartado 3.2.3.4.

- Por último, la aplicación deberá poder cifrar un conjunto de datos antes de que la tarjeta envíe estos datos al terminal. Este cifrado se realizará usando la clave de sesión calculada previamente[21].

5.3.2 Diseño

La ventaja que presenta el uso de Java Card para el desarrollo de aplicaciones monedero en tarjetas inteligentes, es que la API de Java Card proporciona un extenso conjunto de librerías que sirven de apoyo para implementar los mecanismos de seguridad de la tarjeta, mediante métodos capaces de realizar los algoritmos criptográficos necesarios.

Las librerías usadas de Java Card para implementar estas funciones criptográficas son *javacard.security* y *javacardx.crypto* que proporcionan el soporte necesario para desarrollar los mecanismos de seguridad de la aplicación.

Para el cálculo de claves en la tarjeta, se usará una clave administrativa, por lo tanto, se declara un *array global* que contiene los 16 bytes correspondientes a la clave administrativa, y que es accedido para realizar los cálculos de la clave de sesión. En consecuencia, el valor de esta clave administrativa no podrá ser modificado por el usuario, dado que el usuario no accede a este *array* en ningún momento.

Sin embargo, el usuario o el terminal deben conocer el valor de la clave administrativa para poder realizar también el cálculo de la clave de sesión. Este valor se proporcionará al usuario, pero no a través de la propia tarjeta.

Se declaran también globalmente el atributo de tipo DESKey, que almacenará la clave de sesión cada vez que se calcule y el encriptador de tipo Cipher, que se usará siempre que se cifren unos datos o se calcule el algoritmo Triple DES (Hassler V., 2002). Estos atributos se inicializan también por medio de los constructores de la clase que compone la aplicación.

A continuación se analizan los diferentes casos en los que se implementan los mecanismos de seguridad de la tarjeta.

5.3.2.1 Cálculo de la clave de sesión

El cálculo de la clave de sesión se hace durante la ejecución del comando *Internal Authenticate*, pero para este cálculo no se pueden utilizar directamente los métodos proporcionados por Java Card, sino que se realiza un proceso de desafío-respuesta que no se detalla en este proyecto por razones de confidencialidad.

Tanto la tarjeta como el terminal calculan la clave de sesión que se usará para firmar y cifrar los datos que se utilicen en la comunicación.

Esta clave de sesión se almacena en la aplicación mientras se cumplan las condiciones de seguridad y no se desconecte la tarjeta.

[21] Véase apartado 3.2.3.2.

5.3.2.2 Comandos con mensajes securizados

En la aplicación se implementan varios comandos que usan mensajes securizados, como se explica en el apartado 5.4.3. Estos comandos utilizan una clave de sesión calculada previamente, y con ella se realizan los procesos de firmado y cifrado/descifrado de los datos.

Para el cálculo de la firma y del cifrado CBC en lenguaje Java Card se emplea el método *obtener_datos*. Este método se ha desarrollado de manera que, a partir de un *array* de datos, sea capaz de descifrar los datos que contiene y calcular el resultado de la firma para compararla con la recibida. Se utiliza la clave de sesión obtenida anteriormente, y se aplican los métodos de firmado y descifrado sobre los datos que se pretenda firmar o descifrar.

Hay que tener en cuenta que la comparación de la firma se hace únicamente con los tres bytes menos significativos de ésta, y que la tarjeta debe devolver al terminal los tres bytes más significativos de dicha firma.

El método *obtener_datos* devuelve un *array* de bytes con los datos resultantes descifrados, de manera que, posteriormente, esos datos puedan ser manejados por los distintos comandos para realizar operaciones en el sistema de ficheros de la tarjeta. La principal ventaja de este método es que puede ser aplicado a cualquier comando, ya que devuelve los datos en claro y con la misma configuración con la que se reciben en los comandos que no utilizan mensajes securizados.

Para obtener los datos en claro mediante el método *obtener_datos,* se aplica en primer lugar el descifrado sobre los datos recibidos, obviando los tres últimos bytes correspondientes a la firma. Posteriormente, sobre los bytes en claro se calcula la firma incluyendo en el cálculo los bytes de cabecera y, si la comprobación de los tres últimos bytes de la firma coinciden con los últimos tres bytes recibidos, el método *obtener_datos* devuelve los datos en claro.

Cabe destacar que en la estructura de la aplicación se diferencian los métodos que implementan mensajes securizados de los que no los implementan, a pesar de que se trate del mismo comando (en un caso con SM y en el otro sin SM). Por ejemplo, en el caso del comando *Update Binary*, se han desarrollado dos métodos: uno si el comando se usa con mensajes securizados y el otro si se usa el comando en la forma estándar.

Realmente los dos métodos del mismo comando (con SM y estándar) son muy similares, siendo las únicas diferencias el cifrado/descifrado y la firma de los datos; es decir, que los métodos con SM llaman al método *obtener_datos,* mientras que los estándar no lo hacen.

La razón de utilizar dos métodos diferentes se debe a que el primer byte de la cabecera de los comandos con SM (el byte CLA) es distinto en los dos comandos, lo que provoca que, desde el principio de las comprobaciones llevadas a cabo en la tarjeta, la aplicación distinga los procesos de los dos comandos entre sí.

5.3.2.3 Descifrado

Debido a la configuración que se establece del sistema de seguridad, el terminal siempre cifra y la tarjeta siempre descifra. Por esta razón en la aplicación únicamente se usa un método para descifrar los datos recibidos, utilizando la clave de sesión calculada previamente.

El método empleado se denomina *descifrar* y devuelve un *array* de bytes que contiene los datos en claro a partir de los datos cifrados que se introducen como parámetro, aplicando el método de *zero-padding* en caso de ser necesario.

El funcionamiento consiste en dividir los datos recibidos en grupos de 8 bytes (usando *zero-padding* en caso de ser necesario) y aplicar consecutivamente el descifrado de los datos según se muestra en la Figura 10.

```
Util.arrayCopy(datosa, (short)(8*(i-1)), salida2, (short)0, (short)8);
cipherCBC.init(deskey, Cipher.MODE_DECRYPT,salida2,(short)0,(short)8);
cipherCBC.doFinal(datosa,(short)(8*i), (short)8, salida1, (short)0);
Util.arrayCopy(salida1, (short)0, datos_denc, (short)(8*i), (short)8);
```

Figura 10 – Método de descifrado

Los datos de inicialización del descifrado de un grupo de datos de 8 bytes es el grupo de 8 bytes resultante del grupo anterior de datos y, así consecutivamente, hasta el descifrado de todos los datos y comenzando por un bloque de inicialización de todo ceros[22].

A este método se llama desde *obtener_datos* de manera que se puedan descifrar los datos y adecuarlos a la presentación necesaria para ejecutar los distintos comandos que implementa la aplicación.

5.3.2.4 Firmado

El método de firmado es fundamental para la implementación de los mecanismos de seguridad de la aplicación, ya que la firma permite comprobar la identidad del terminal y de la tarjeta con el conocimiento de la *clave administrativa* y mediante el uso de la *clave de sesión* calculada a partir de ésta.

El método de firmado en la aplicación se denomina *firmar* y realiza un proceso similar al método *descifrar*. En este método se dividen también los datos que se han de firmar en grupos de 8 bytes, aplicando el método *zero-padding* en caso de ser necesario, y aplican los métodos de firmado a los bloques de 8 bytes sucesivamente, utilizando el resultado del firmado del bloque anterior. Pero el resultado se almacena en el bloque de 8 bytes final, al cual se le aplica por último el algoritmo *TripleDES*[23] como se muestra en la Figura 11.

[22] Véase apartado 3.2.3.2.
[23] Véase apartado 3.2.3.4.

```
for(int i=1;i<veces;i++){

    cipherCBC.init(skX, Cipher.MODE_ENCRYPT,salida1,(short)0,(short)8);
    cipherCBC.doFinal(datosa,(short)(8*i), (short)8, salida1, (short)0);


}

skX.setKey(sk2, (short)0);
cipherCBC.init(skX, Cipher.MODE_DECRYPT,new byte[]{0,0,0,0,0,0,0,0},(short)0,(short)8);

cipherCBC.doFinal(salida1,(short)0, (short)8, salida1, (short)0);

skX.setKey(sk1, (short)0);
cipherCBC.init(skX, Cipher.MODE_ENCRYPT,new byte[]{0,0,0,0,0,0,0,0},(short)0,(short)8);

cipherCBC.doFinal(salida1,(short)0, (short)8, datos_fir, (short)0);
```

Figura 11 – Método de firmado

5.4 Comandos Implementados

En este apartado se tratan los comandos implementados en el proyecto, explicando los requisitos de cada comando y la implementación efectuada mediante tecnología Java Card para lograr la emulación de la tarjeta WG10.

Sin embargo, por razones de confidencialidad, en este proyecto no se detallan los valores de los campos ni la codificación exacta de los comandos implementados, si no que se hace un breve resumen de cada uno de los comandos.

5.4.1 Comandos de Administración de Ficheros

En este apartado se agrupan los comandos encargados de la administración del sistema de ficheros, es decir, los comandos que se dedican a crear, modificar y acceder a los ficheros.

5.4.1.1 Create File (Estándar)

Este comando será el utilizado para crear ficheros EF o DF en el directorio que se haya seleccionado previamente. La atribución de "Estándar" indica que este comando en concreto se ejecuta sin usar mensajes securizados.

En el caso del comando *Create File*, la información del fichero consiste en una secuencia de bytes que, según su codificación establecerán las características que tendrá el fichero creado. Estos parámetros para el comando *Create File* son:

- **Identificador del fichero**
- **Tipo de fichero**

- **Byte de opciones del fichero**
- **Tamaño del fichero**
- **Condiciones de acceso en actualización**
- **Condiciones de acceso en lectura**[24]

Según resulte la ejecución de este comando, la tarjeta debe devolver una palabra de dos bytes que indica el estado en que se encuentra la tarjeta después de la implementación del comando.

Cabe señalar que con la creación del fichero no se escribirá ningún dato en el *array de memoria* destinado a guardar los datos contenidos en los ficheros, ya que a este *array* sólo se accederá al utilizar los comandos de escritura en fichero[25].

5.4.1.2 Select File

Este comando se utiliza para seleccionar un fichero dentro de un directorio de la tarjeta inteligente. Su funcionamiento implementado en tecnología Java Card consiste en recorrer el *array* que guarda el direccionamiento de la memoria de la tarjeta y en comprobar el identificador de todos los ficheros almacenados en dicho *array* hasta dar con el identificador especificado en el comando, en ese momento se leerá el resto de la información sobre el fichero almacenada en el *array de direccionamiento* y la aplicación será capaz de realizar operaciones sobre el fichero seleccionado.

5.4.1.3 Read Binary

El comando *Read Binary* lee los datos contenidos en un fichero de tipo EF transparente binario una vez especificado el byte desde el cual se desea empezar a leer el fichero y el número de bytes que se han de leer.

Este comando admite selección implícita de ficheros, lo que significa que, si se utiliza el modo de selección implícita de este comando, no es necesario haber ejecutado anteriormente el comando *Select File* para seleccionar el fichero sobre el que implementar el comando. Esta selección implícita se codifica según los valores que toman los campos P1 y P2 del comando.

5.4.1.4 Update Binary

La ejecución de este comando es muy similar a la que lleva a cabo el comando *Read Binary,* la principal diferencia entre estos dos comandos radica en que *Update Binary* sobrescribe los datos existentes en el fichero seleccionado en un offset indicado.

[24] El análisis de las condiciones de acceso en lectura y en actualización de los ficheros se ha explicado en el apartado 5.3.
[25] Véase apartado 5.2.

Al igual que el comando *Read Binary*, este comando también presenta la opción de selección implícita del fichero que se ha de actualizar. Esta selección implícita tiene en los dos casos la misma configuración de los campos P1 y P2.

La implementación de este comando sigue unos pasos similares a la del comando *Read Binary*, llevando a cabo la selección implícita de fichero, si es necesario, e identificando los bytes a modificar dentro del fichero seleccionado, para, posteriormente, sustituirlos por los recibidos en el campo de datos.

5.4.1.5 Append Record

El comando *Append Record* crea un registro al final de un fichero con estructura de registros. Un fichero con estructura de registros consiste en un número determinado de registros de tamaño fijo. Sin embargo, al crear el fichero mediante el comando *Create File*, se define el tamaño del fichero y el tamaño de los registros que almacenará (de lo que se puede obtener el número de registros del fichero), pero se debe inicializar cada registro y su posición en el fichero antes de leerlo o de actualizarlo, y esto se hace mediante el comando *Append Record.*

En el caso de ficheros con estructura de registros, se puede efectuar también una selección implícita del fichero; sin embargo, esta selección se lleva a cabo de forma distinta a la que se realiza para los ficheros trasparentes, ya que únicamente se utiliza el byte del campo P2.

5.4.1.6 Update Record

El comando *Update Record* actualiza los datos que se encuentran en un registro dentro de un fichero con estructura de registros. Este comando también tiene la opción de implementarse con selección implícita de fichero mediante el uso del byte del campo P2.

Cabe señalar que antes de poder ejecutar el comando *Update Record*, se ha debido ejecutar el comando *Append Record*, ya que, en caso contrario, la tarjeta devolverá la excepción correspondiente al no estar inicializado el registro sobre el que se quiere escribir.

5.4.1.7 Read Record

Este comando tiene un funcionamiento similar al del comando *Update Record*. La diferencia está en que en el caso de *Read Record* la tarjeta devuelve los bytes de los que consta el registro seleccionado.

La implementación de este comando varía con respecto a la del comando *Update Record* en que, en este caso, los datos del registro seleccionado que se han de leer, son devueltos por la tarjeta como resultado de las operaciones realizadas.

5.4.2 Comandos de Seguridad

Con el objetivo de implementar las funcionalidades de las que dispone un monedero electrónico, la tarjeta WG10 define ciertos comandos que controlarán las condiciones de acceso a los ficheros y generarán y comprobarán las claves necesarias para guardar el cumplimiento de la seguridad necesaria en una tarjeta monedero.

5.4.2.1 Internal Authenticate

El comando *Internal Authenticate* tiene gran relevancia en la seguridad de la tarjeta, ya que establece una clave de sesión válida en el directorio actual.

Este comando incrementa el número de transacciones, y a partir de una clave administrativa, genera una clave de sesión temporal. Asimismo calcula una firma a partir de un reto recibido del terminal y proporciona la firma para la autenticación de la tarjeta, para, finalmente, calcular la clave de sesión para el directorio.

El reto del terminal consiste en un conjunto de bytes a partir de los cuales el comando efectuará los cálculos necesarios para generar la clave de sesión.

En caso de que el comando concluya correctamente, la tarjeta deberá enviar una respuesta al terminal con la firma realizada, para que el terminal pueda calcular a su vez la clave de sesión y poder realizar la autenticación.

5.4.2.2 Set Access Conditions

La función de este comando es establecer las reglas de acceso a un fichero en lectura o en actualización. Para que este comando se pueda ejecutar sobre un fichero determinado, debe satisfacer la condición de acceso en actualización definida por el directorio actual.

Durante la ejecución de *Set Access Conditions*, es necesario comprobar también la compatibilidad de las condiciones de acceso antiguas y nuevas en el fichero.

El comando *Set Access Conditions* está implementado en la aplicación de manera que sea capaz de evaluar las condiciones de acceso del fichero, comparar la compatibilidad con las nuevas condiciones que se quieren implantar y cambiarlas en caso de que la comparación de un resultado permitido[26].

5.4.2.3 Get Response

El comando *Get Response* permite al terminal obtener datos que envía la tarjeta como respuesta a otros comandos ejecutados anteriormente.

En ocasiones la respuesta de la tarjeta tras la ejecución de los comandos consta de más bytes de los dos normalmente usados denominados *Status Word* (SW). Por ejemplo, en el caso de que el comando *Select File* sobre un DF concluya correctamente, la tarjeta responde:

[26] Las condiciones de acceso a los ficheros y su compatibilidad se han tratado de una manera más extensa en el apartado 5.3.

SW=61, 14; lo que indica que tiene 20 bytes a transmitir como resultado del comando.

La existencia de *Get Response* se explica por las limitaciones que presenta el protocolo T=0, que no permite recibir y devolver datos en el mismo comando. *Get Response* hace posible que se devuelvan los datos generados con el comando inmediatamente anterior.

Si después de una operación de monedero la tarjeta no recibe un *Get Response*, la transacción será cancelada y los datos del fichero restaurados. Esta participación en el mecanismo de invisibilidad de las transacciones monedero hace clasificar a *Get Response* dentro del grupo de comandos de seguridad, a pesar de que es utilizado también en comandos de administración de ficheros.

Tras la ejecución del comando *Get Response* la tarjeta devolverá los datos solicitados completos o, en caso de no poder enviarlos completos, parte de los datos, con una palabra estado que indique el número de bytes que quedan por leer.

En la implementación de *Get Response* mediante tecnología Java Card, se utiliza un *array* especialmente para almacenar la información generada que se obtendrá después por este comando.

5.4.2.4 Verify Secret Code

El comando *Verify Secret Code* compara un código secreto recibido del terminal con un código secreto almacenado en la tarjeta. En caso de fallo en los valores del código la tarjeta devuelve el número de intentos restantes. También se puede obtener el número de intentos restantes enviando el mismo comando con el campo de datos vacíos (sin introducir el código secreto).

Una vez satisfechas las condiciones necesarias, este comando accede al número secreto almacenado en la tarjeta y lo compara con el recibido desde el terminal; si coinciden, el comando finaliza correctamente.

5.4.3 Mensajes Securizados

La tarjeta WG10 soporta el envío y recepción de mensajes securizados (SM) y, por tanto, la emulación a partir del *applet* Java Card también identifica y ejecuta los comandos con mensajes securizados.

Los comandos implementados con mensajes securizados tienen las mismas funcionalidades que los comandos, con el mismo nombre, que no requieren mensajes securizados. Pero, en este caso, la información enviada por el terminal en el campo de datos y la enviada desde la tarjeta al terminal se encriptan con la clave de sesión calculada, después de haber calculado la firma correspondiente a los datos para la transmisión por SM.

La característica que identifica a los comandos con mensajes securizados es que su byte de clase será **CLA=04** en vez de **CLA=00** ó **CLA=84** en vez de **CLA=80**.

El proceso para la codificación de la transmisión con mensajes securizados, tanto en la tarjeta como en el terminal, es el siguiente:

1. Se calcula la clave de sesión mediante el comando *Internal Authenticate*[27]. Se calcula la firma de todos los datos que se van a transmitir, esto es, se incluyen en este cálculo también los campos de la cabecera del comando.
2. Una vez se ha realizado el cálculo de la firma[28], se cogen los 3 bytes menos significativos del resultado y se concatenan como los 3 bytes menos significativos a los datos a transmitir que tendrán que ser cifrados.
3. En este punto, se encriptan mediante CBC los datos y se añaden antes de los 3 bytes concatenados de la firma[29]. En este caso se utilizan únicamente los datos del llamado campo de datos de la transmisión.
4. Una vez encriptada, la información se puede enviar el comando con SM.

Cabe señalar que en el caso de una transmisión desde la tarjeta, no se diferencia entre los datos sobre los que calcular la firma y los datos sobre los que calcular el cifrado posteriormente, ya que la tarjeta transmite los datos al terminal sin ninguna cabecera.

Comandos con mensajes securizados:

- ***Update Binary*:** se puede utilizar para la actualización de claves y números secretos, cumpliendo las condiciones de acceso de los ficheros correspondientes. Se emplea también para la actualización de datos sensibles. Cualquier fallo en las condiciones de acceso provoca un fallo en la tarjeta y el borrado de la clave de sesión.
- ***Update Record:*** para que el comando sea procesado, se deben cumplir todas las condiciones de acceso al fichero de registros y debe seleccionarse correctamente el fichero y el registro sobre el que realizar la operación. En caso contrario se abortará el comando y se borrará la clave de sesión.
- ***Append Record:*** el comando sólo será procesado si se cumple la condición de acceso en actualización del fichero. El resto de operaciones sobre registros únicamente se podrá llevar a cabo, si se ha creado previamente el registro mediante *Append Record.*
- ***Create File:*** efectúa la operación de crear un fichero en el directorio actual de forma segura, utilizando propiedades criptográficas. Al usarse mensajes securizados, la clave de sesión ha tenido que ser calculada anteriormente para evitar que el comando sea rechazado. Este es un caso especial, ya que si en el directorio actual no se encuentra la clave administrativa o el fichero de número de transacción, se permitirá usar la clave del MF (directorio maestro).
- ***Set Access Conditions:*** este comando establece las reglas de acceso a un fichero en lectura o en actualización, usando propiedades criptográficas. El análisis de las condiciones de acceso a los ficheros se detalla en el apartado 5.3. En caso de que no se cumplan estas condiciones o el comando resulte en algún error, se abortará la operación y se borrará la clave de sesión.

[27] Véase apartado 5.4.2.1.
[28] Véase apartado 3.2.3.4.
[29] Véase apartado 3.2.3.2.

Los comandos que utilizan mensajes securizados son diferentes a los comandos estándar con el mismo nombre, y como tales están desarrollados de forma independiente en la aplicación. Todo ello a pesar de que las implementaciones del mismo comando con SM y sin ella sean muy similares, dado que efectúan las mismas operaciones una vez pasado el proceso de firmado y cifrado/descifrado que realizan los comandos con SM.

Esto es necesario debido a que los protocolos que definen el funcionamiento de la tarjeta WG10 y de las tarjetas inteligentes de monedero electrónico, especifican exactamente en qué orden se deben analizar los datos, es decir, primero se comprueba el campo CLA, luego el INS, y así sucesivamente.

La posibilidad del uso de mensajes securizados en tarjeta, es una de las principales características que permite utilizar esta tarjeta o la aplicación que se desarrolla en este trabajo, como monedero electrónico.

6 Herramientas de Desarrollo

6.1 Developer Suite

6.1.1 Introducción

El entorno de desarrollo o IDE utilizado para este proyecto es el Developer Suite de Gemalto, una empresa dedicada a la seguridad digital que proporciona herramientas para desarrolladores y también soluciones para sectores como la banca, las administraciones públicas o las empresas. La tarjeta inteligente utilizada para este trabajo también está fabricada por Gemalto.

Developer Suite es, por tanto, un Integrated Development Enviroment (IDE)[30] desarrollado por Gemalto basado en Eclipse[31], que permite programar aplicaciones para tarjetas inteligentes Java Card en un entorno muy similar a los utilizados para las aplicaciones Java convencionales, proporcionando además *wizards* o asistentes que ayudan al usuario a definir los estándares de las aplicaciones que va a desarrollar.

Además, Developer Suite proporciona un conjunto de simuladores para realizar las pruebas de funcionamiento de las aplicaciones desarrolladas, estando disponibles simuladores para tarjetas SIM, USIM y R-UIM, o para estándares de comunicaciones móviles como 2G, 3G o CDMA.

En la actualidad Gemalto pone a disposición de los desarrolladores la última versión del entorno (Developer Suite 3.6.2) y también una versión de prueba de 20 días. Sin embargo, para este trabajo se ha utilizado una versión anterior (Developer Suite 3.4.2) debido al alto coste de la última versión.

6.1.2 Programación de applet Java Card

Para la programación del *applet* de este proyecto, se ha hecho uso de los *wizards* disponibles en el entorno de desarrollo, de manera que sea más sencilla la definición de los estándares de la tarjeta, cumpliendo con las especificaciones del ETSI y de Global Platform.

En primer lugar se debe seleccionar el *wizard* correspondiente a la creación de un nuevo proyecto Java Card, siguiendo los pasos que indica el asistente y definiendo el nombre y el Application Identifier (AID) del paquete.

[30] Un IDE es un software que proporciona facilidades para el desarrollo como edición del código, compilación o depuración.
[31] Eclipse es un IDE de código abierto creado por la iniciativa Eclipse.

En el punto en el que el asistente propone elegir el tipo de tarjeta para la que se diseñará el *applet,* se debe escoger la opción USim Card R6 (como se muestra en la Figura 12), con lo que el proyecto creado cumplirá con los estándares necesarios.

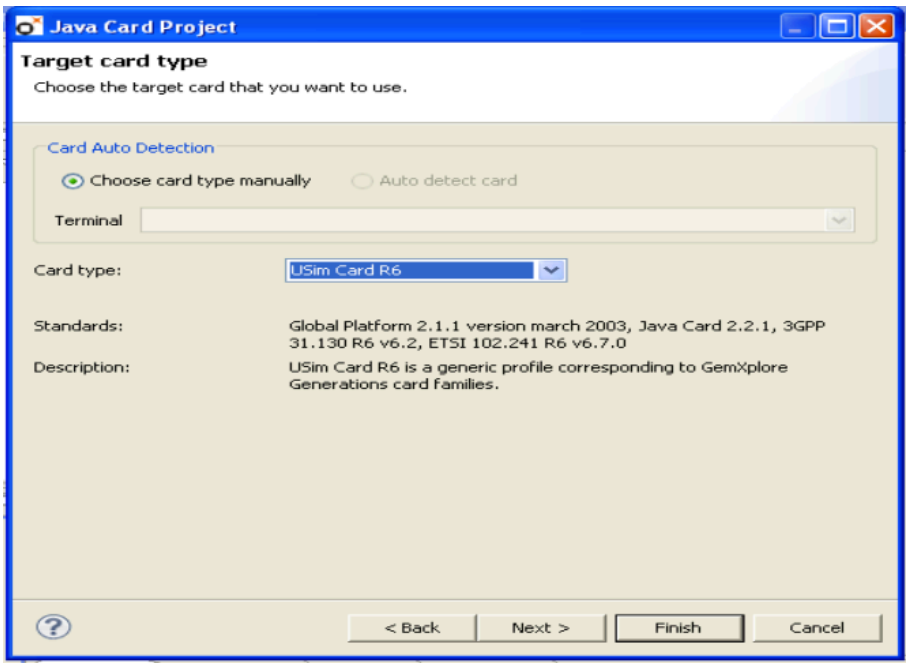

Figura 12 – Configuración de Developer Suite

Posteriormente se seleccionará también la herramienta JCardManager que será la responsable de la simulación y la instalación de la aplicación en la tarjeta.

Una vez creado el proyecto, se selecciona el *wizard* correspondiente a la creación de un *Applet Java Card*, siguiendo los pasos del asistente y configurando el AID[32] de la aplicación y seleccionando las especificaciones de Global Platform.

El AID que se defina para la aplicación tendrá gran importancia, ya que será necesario para seleccionar dicha aplicación dentro de la tarjeta de la misma forma que se selecciona un fichero DF por nombre. En una tarjeta Java Card el fichero maestro o MF lo compone el directorio principal de la tarjeta. Después se pueden seleccionar los distintos DF que serán las distintas aplicaciones instaladas en la tarjeta, pudiendo tener cada aplicación una funcionalidad distinta y teniendo necesariamente un AID único.

Una vez dados estos pasos, ya se puede empezar a desarrollar el *applet* que se instalará en la tarjeta.

[32] El AID es el identificador único de la aplicación Java Card y consta de 16 bytes.

6.1.3 Depuración y compilación

El IDE Developer Suite proporciona también posibilidades de depuración y compilación del código de manera que se puedan resolver los posibles errores y se pueda generar un archivo .cap para instalarlo posteriormente en la tarjeta inteligente Java Card.

El propio Developer Suite proporciona un asistente similar al que tienen otros IDE que detecta los posibles errores en el código y presenta, como sugerencias, diferentes soluciones a estos errores.

Por otro lado también es posible establecer varios puntos de parada a lo largo del código, de manera que, una vez compilado el código y usando el simulador, se pueda analizar el funcionamiento de la aplicación y depurar los errores en el código.

Tras haber compilado el código, para la simulación y la instalación del *applet* en la tarjeta se usará la herramienta JCardManager, que ya viene proporcionada en el Developer Suite 3.4.2.

6.2 JCardManager

6.2.1 Introducción

JCardManager es una herramienta que proporciona Gemalto para simular e instalar aplicaciones siguiendo las especificaciones de Java Card y Global Platform. Permite realizar las pruebas necesarias con los *applets* desarrollados y se integra en Developer Suite de manera que se puedan probar las aplicaciones programadas en este IDE.

Dentro de las opciones de Developer Suite es posible seleccionar otros simuladores distintos, sin embargo, JCardManager proporciona los estándares y las especificaciones de Global Platform necesarias para la aplicación de este proyecto.

6.2.2 Simulador

El simulador que implementa JCardManager permite enviar y recibir comandos imitando el comportamiento y los protocolos de una tarjeta inteligente, ejecutando el código antes programado y compilado en el Developer Suite, de manera que se puedan hacer las pruebas necesarias para la comprobación del funcionamiento de la aplicación.

Para abrir JCardManager se selecciona del menú desplegable disponible en el IDE Developer Suite.

Una vez abierto JCardManager, la simulación se llevará a cabo utilizando las opciones proporcionadas en el menú desplegable correspondiente al tipo de tarjeta que se utiliza (en este caso Usim Card R6), de la forma que se muestra en la Figura 13.

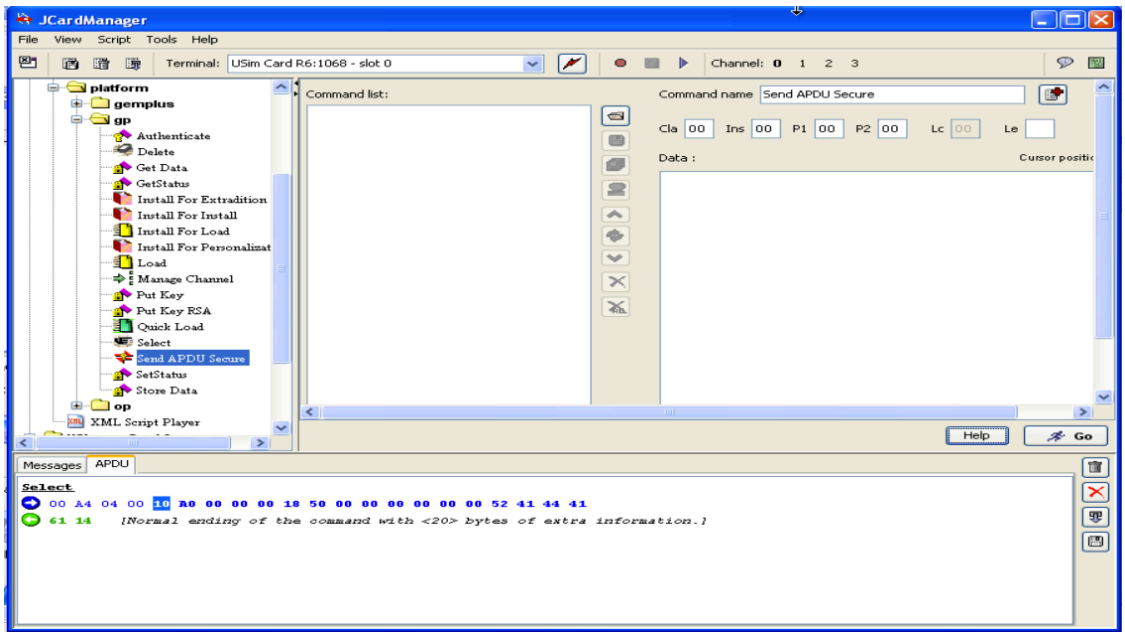

Figura 13 – Configuración de JCardManager

Para seleccionar el AID del *applet* programado, se utiliza la opción *Select*, introduciendo el AID deseado, tras lo cual se podrán enviar los comandos que se deseen a la tarjeta para ser ejecutados por la aplicación desde la opción *Send APDU Secure*, apareciendo en la ventana inferior los comandos enviados y la respuesta de la tarjeta como se muestra en la Figura 13.

Gracias a esta funcionalidad de JCardManager, es posible realizar las pruebas de las diferentes funcionalidades de la aplicación sin necesidad de instalarla en la tarjeta, y haciendo uso del depurador que proporciona el IDE Developer Suite para reparar los posibles errores.

6.2.3 Instalación en la tarjeta

La instalación de *applets* en la tarjeta también se llevará a cabo mediante JCardManager, ya que permite gestionar los archivos que contenga la tarjeta, así como instalar o eliminar aplicaciones contenidas en ella.

Sin embargo, antes de realizar alguna de estas operaciones, es necesario autenticarse, a través de la opción *Authenticate,* mediante el uso de un fichero de claves *.keys* creado previamente[33] y una vez esté introducida la tarjeta en el lector y éste conectado al ordenador de manera que JCardManager reconozca la tarjeta y pueda realizar operaciones sobre ella.

[33] El fichero de claves se puede crear fácilmente usando el propio JCardManager.

Para la instalación del *applet* en la tarjeta se selecciona la opción *Quick Load*, se introduce el AID de la aplicación a instalar y se selecciona el archivo *.cap* resultante de la compilación de la aplicación. Será necesario también incluir el archivo de claves para la instalación del *applet* en la tarjeta que autentificará dicha instalación.

Una vez realizados estos pasos, el *applet* estará instalado en la tarjeta y ésta se podrá utilizar ejecutando las funcionalidades de la aplicación instalada, siempre y cuando se seleccione previamente el AID correspondiente.

7 Pruebas de Funcionamiento

En este capítulo se documentan las pruebas de funcionamiento realizadas con la aplicación, mostrando las respuestas de ésta ante los diferentes comandos de entrada y analizando las funciones de gestión del sistema de ficheros y del sistema de seguridad de la tarjeta.

Los resultados se muestran en imágenes capturadas durante las pruebas de la tarjeta.

7.1 Pruebas del sistema de ficheros

7.1.1 Seleccionar fichero DF

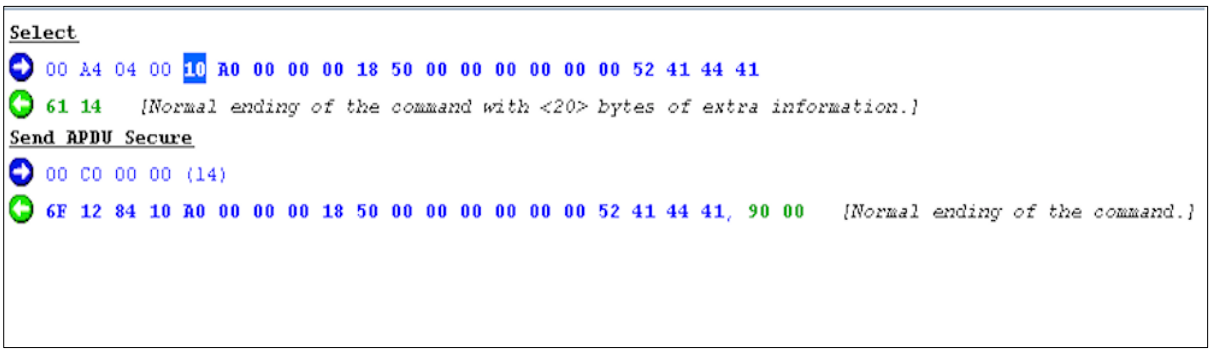

Figura 14 – Prueba seleccionar DF

En esta captura se muestra la respuesta de la tarjeta ante la recepción del comando *Select* sobre un fichero de tipo DF y el comando *Get Response* para obtener la respuesta a esta selección.

La tarjeta selecciona correctamente el fichero correspondiente, e indica al terminal que tiene preparados 20 bytes para enviar. Por esta razón en la captura se muestra también la ejecución del comando *Get Response* y la respuesta de la tarjeta a ese comando, que corresponde a los 20 bytes de información del fichero.

Al tratarse de una tarjeta Java Card, esta selección de DF corresponde realmente a la selección de la aplicación dentro de las aplicaciones que contiene la tarjeta, y la respuesta ha sido codificada especialmente para emular la respuesta que ofrecería una tarjeta WG10.

7.1.2 Crear fichero lineal transparente

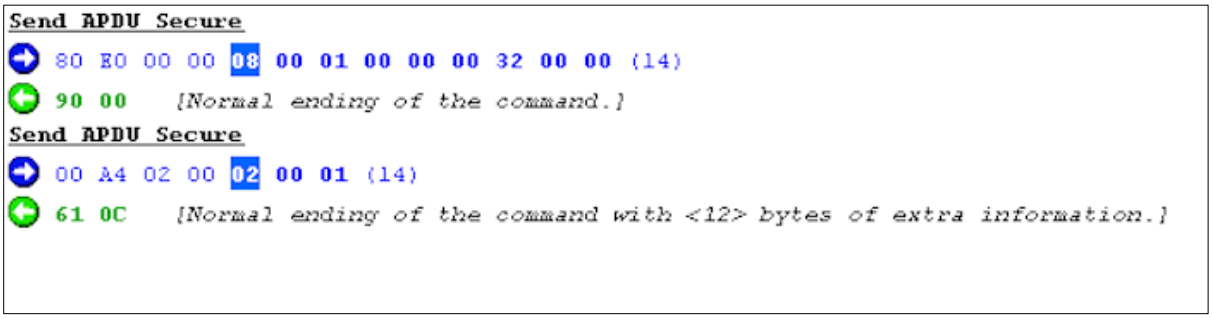

Figura 15– Prueba crear fichero transparente

En esta captura aparece en primer lugar el comando correspondiente a la creación de un fichero transparente binario mediante el comando *Create File,* con la respuesta de la correcta creación del fichero por parte de la tarjeta.

En la captura se muestra también la selección del fichero creado mediante el uso del comando *Select File*. La respuesta de la tarjeta indica también la correcta selección del fichero, de manera que se ha encontrado y almacenado su información, entre la que está su posición, en memoria y que será útil para posibles usos posteriores.

7.1.3 Escritura y lectura estándar

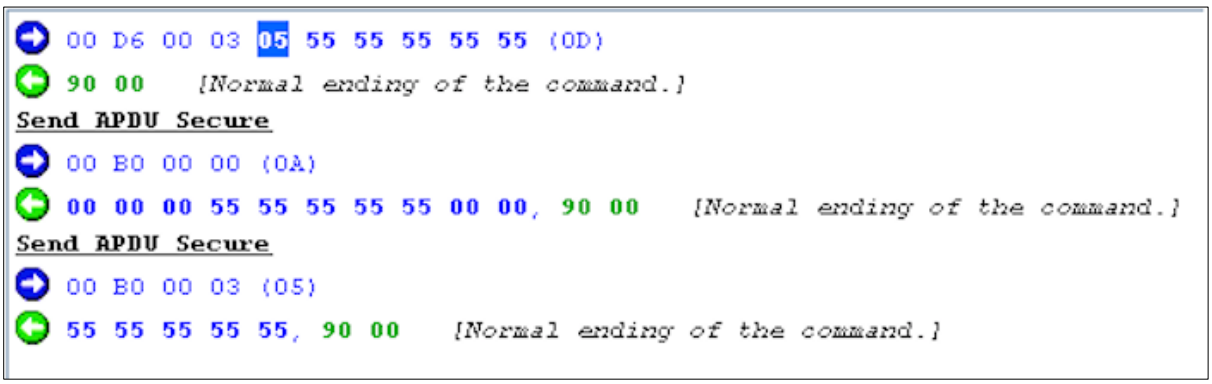

Figura 16 – Prueba lectura y escritura de EF

Como el fichero creado ya se ha seleccionado anteriormente, en esta captura se muestra la escritura de 5 bytes en el fichero EF mediante el uso del comando *Update Binary*, con el valor 55 en hexadecimal y almacenados a partir de la cuarta posición del fichero.

Posteriormente, en la captura aparecen dos ejecuciones del comando *Read Binary* aplicado sobre el fichero. En la primera se leen los 10 primeros bytes del fichero y, como se puede observar, se comprueba que se han actualizado al valor 55 los 5 bytes a partir de la cuarta posición.

En la segunda ejecución del comando *Read Binary,* se escoge leer únicamente los 5 bytes actualizados mediante la correcta codificación de los parámetros de este comando.

7.1.4 Escritura y lectura implícitas

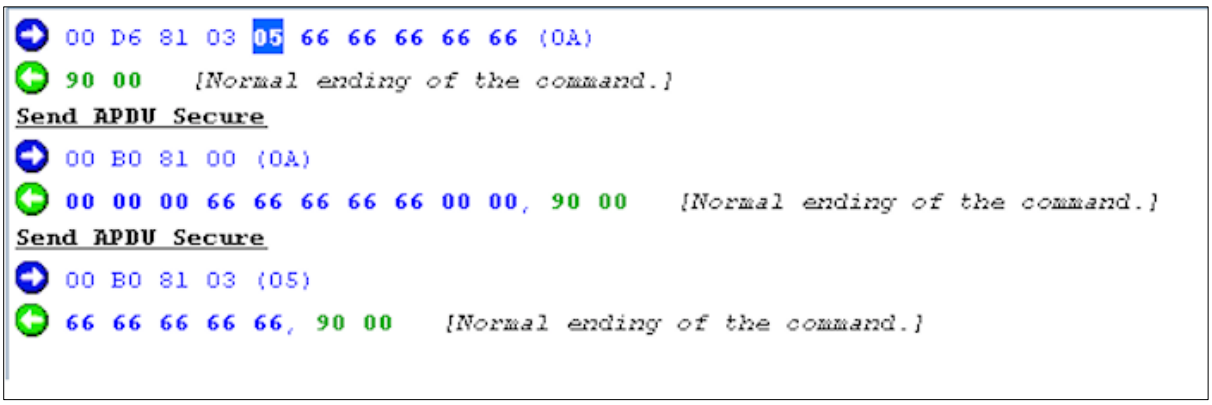

Figura 17 – Prueba escritura y lectura implícitas de EF

En esta captura se muestra la actualización y lectura mediante selección implícita del EF creado anteriormente. Gracias a esta selección implícita no es necesario haber ejecutado el comando *Select File* para poder actualizar o leer el fichero.

El primer comando que se muestra en la captura es *Update Binary* usando selección implícita, escribiendo el valor 66 en cinco posiciones del fichero a partir de la cuarta posición.

El segundo comando mostrado es *Read Binary* aplicado a los 10 primeros bytes del archivo, de manera que se pueden observar las posiciones vacías debido al *offset* configurado en la actualización.

Finalmente se define también el *offset* correspondiente en los parámetros de *Read Binary* para leer únicamente las posiciones deseadas.

7.1.5 Crear fichero con registros

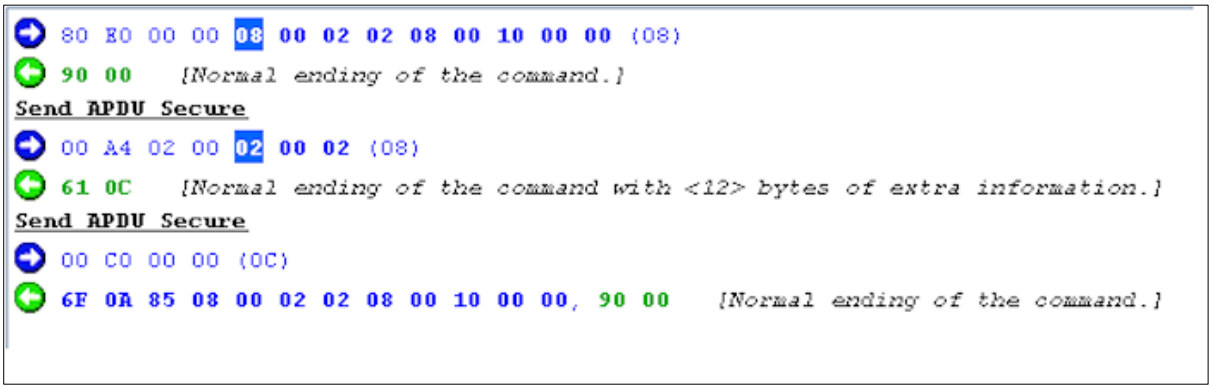

Figura 18 – Prueba crear fichero con registros

En esta captura se muestran los comandos correspondientes a la creación y selección de un fichero con estructura de registros.

En primer lugar se ejecuta el comando *Create File* configurando sus parámetros para crear un fichero con dos registros de longitud 8 bytes.

En segundo lugar se realiza la selección del fichero a través del comando *Select File*. La respuesta de la tarjeta a este comando indica que tiene 12 bytes que enviar al terminal.

Finalmente se ejecuta el comando *Get Response* que devuelve los 12 bytes que envía la tarjeta correspondientes a la información del fichero seleccionado.

7.1.6 Inicializar y leer registros

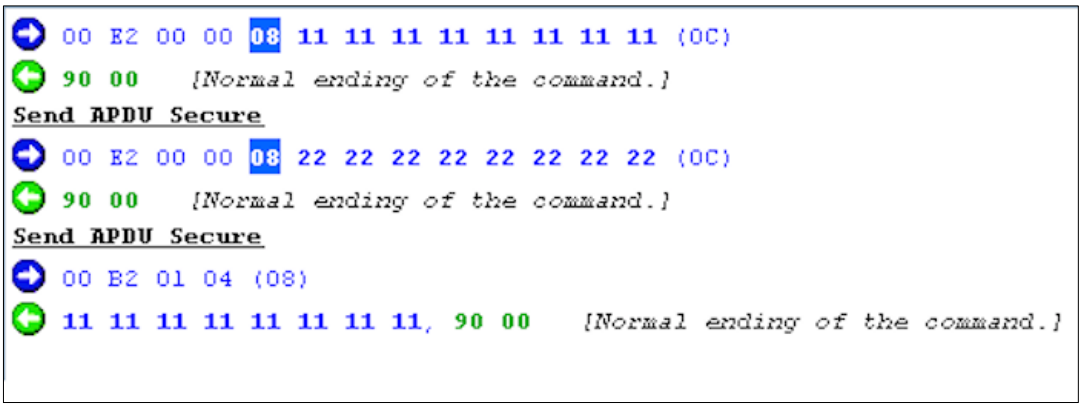

Figura 19 – Prueba inicializar y leer registro

En esta captura se muestran en primer lugar las dos ejecuciones del comando *Append Record,* que inicializa los registros del fichero. El primer registro es inicializado con el valor 11 en hexadecimal en todas sus posiciones y el segundo registro es inicializado con el valor 22 en hexadecimal en todas sus posiciones.

El tercer comando que aparece en la captura corresponde a *Read Record* de tipo estándar que lee todas las posiciones del primer registro del fichero, devolviendo los valores como se observa en la imagen.

7.1.7 Lectura y escritura implícita de registros

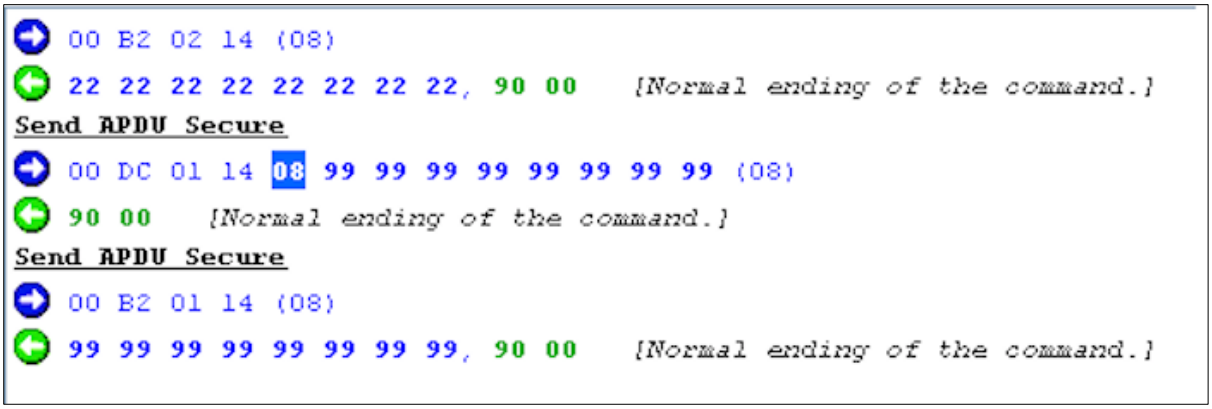

Figura 20 – Prueba lectura y escritura implícitas de registro

En esta captura se muestra en primer lugar la lectura del segundo registro del fichero usando el comando *Read Record* con selección implícita de fichero, devolviendo la tarjeta los valores de los bytes contenidos en el segundo registro (22 en hexadecimal).

Posteriormente se ejecuta el comando *Update Record* usando también selección implícita de fichero y actualizando los 8 bytes del primer registro al valor 99 en hexadecimal.

Por último se leen los datos del primer registro recientemente actualizado, usando *Read Record* con selección implícita, devolviendo la tarjeta los valores del primer registro del fichero.

7.2 Pruebas del sistema de seguridad

7.2.1 Establecimiento de las condiciones de acceso

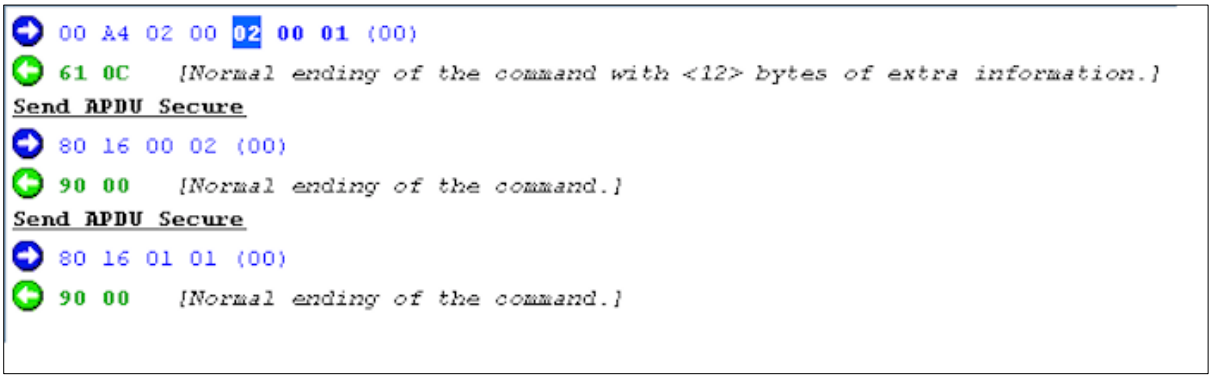

Figura 21 – Prueba establecimiento de condiciones de acceso.

En esta captura se muestra en primer lugar la selección mediante el comando *Select File* del fichero lineal transparente binario creado en el apartado 7.1. Este fichero se creó originalmente con las condiciones de acceso en lectura y escritura libres.

Los dos siguientes comandos de la captura corresponden al uso de *Set Access Conditions* para establecer la condición de actualización por clave secreta, y la condición de lectura por código secreto respectivamente.

7.2.2 Prueba de funcionamiento de las condiciones

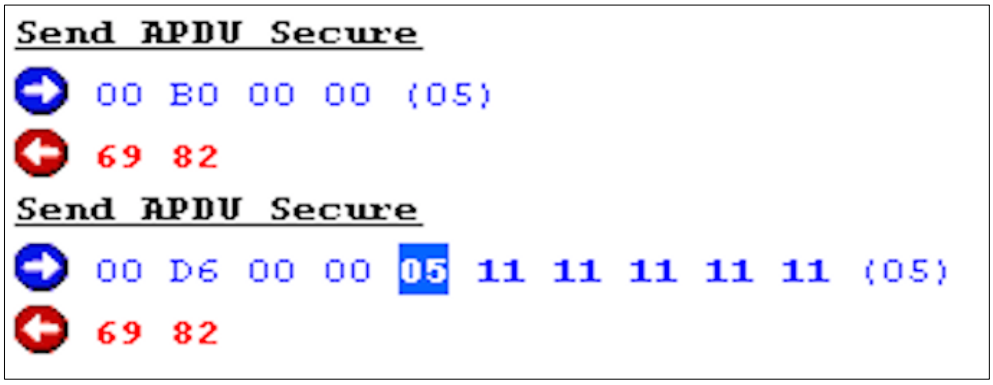

Figura 22 – Prueba comprobación de establecimiento de condiciones

Para comprobar que las condiciones de acceso en lectura y escritura se han establecido correctamente, se prueba, como se muestra en la captura, a leer y a escribir en el fichero.

En primer lugar se prueban a leer los 5 primeros bytes del fichero y en segundo lugar se prueba a escribir el valor 11 en hexadecimal en los 5 primeros bytes del fichero. En los dos casos la tarjeta responde con la excepción correspondiente a que las condiciones de acceso no han sido satisfechas.

7.2.3 Generación de la clave de sesión y actualización segura

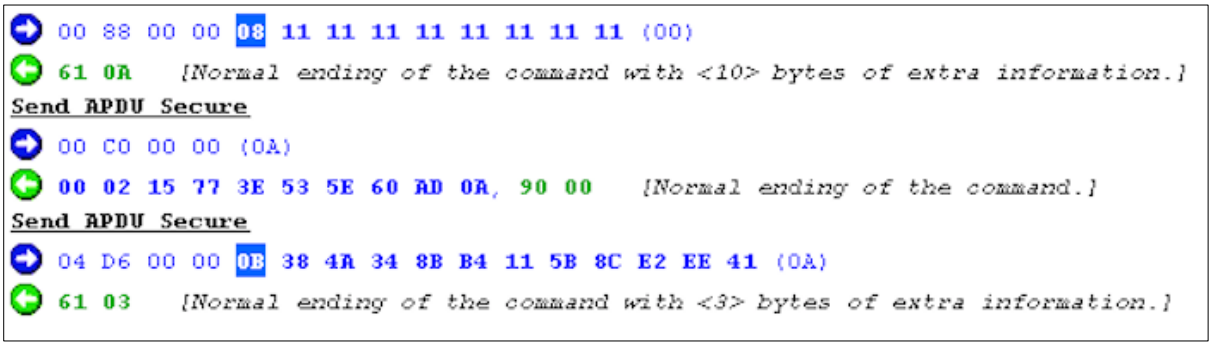

Figura 23 – Prueba generación de clave de sesión

En esta captura se muestra la ejecución del comando *Internal Authenticate* que genera cada vez que se ejecuta una clave de sesión de la tarjeta a partir de un reto enviado por el terminal y el número de transacción (número de veces que se haya ejecutado el comando).

En este caso el reto del terminal consiste en 8 bytes de valor 11 en hexadecimal y el número de transacción es 02.

La tarjeta devuelve una respuesta obtenida mediante el comando *Get Response* que consiste en el número de transacción utilizado y la firma del reto del terminal, datos con los cuales el terminal podrá también calcular la clave de sesión. Por confidencialidad, en este trabajo no se explica el proceso de generación de la clave de sesión ni se entra en detalle sobre todas las características del comando *Internal Authenticate.*

Por último, en la captura se muestra el comando de actualización del fichero (*Update Binary*), usado con mensajes securizados, de manera que los datos a grabar están cifrados y firmados utilizando la clave de sesión calculada. En este ejemplo los datos consisten en 8 bytes de valor 33 en hexadecimal.

Tras la escritura en el fichero, la tarjeta devuelve también una parte de la firma utilizada y de esta manera se puede verificar su autenticidad.

7.2.4 Verificación del código secreto

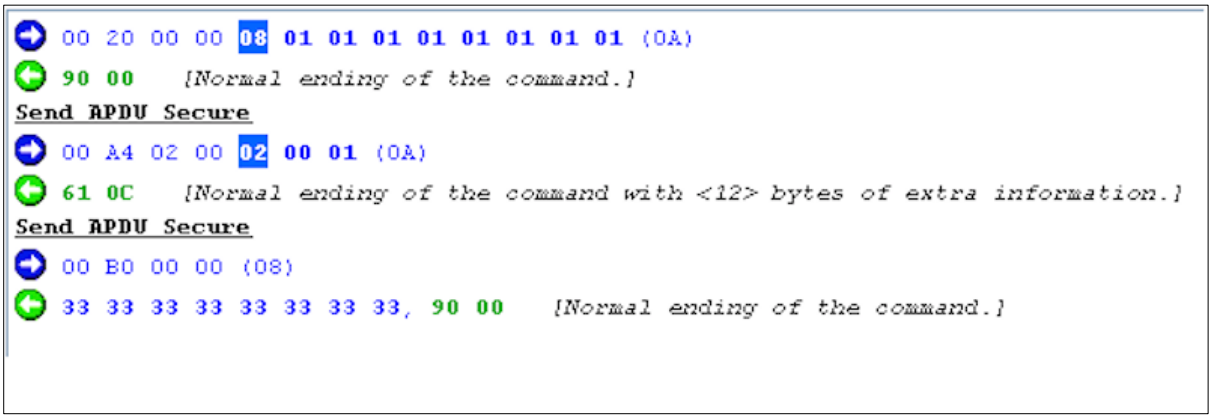

Figura 24 – Prueba verificación del código secreto

El acceso en lectura del fichero está protegido por código secreto, y por esta razón es necesario ejecutar el comando *Verify Secret Code* como se muestra en la captura, siendo el código secreto 8 bytes de valor 01 en hexadecimal.

Tras llevar a cabo la verificación del código secreto en el directorio donde se encuentra el fichero (el DF correspondiente), se procede a leer el fichero mediante el comando *Read Binary*, devolviendo la tarjeta el valor de los datos contenidos en las posiciones del fichero leídas (8 primeros bytes del fichero leídos de valor 33 en hexadecimal).

7.3 Observaciones

En este apartado se exponen ciertas consideraciones sobre la pruebas realizadas con la tarjeta Java Card y el funcionamiento de la aplicación.

- Además de las pruebas explicadas en este capítulo que muestran el correcto funcionamiento de la aplicación, se han realizado todas las comprobaciones correspondientes de cada comando y método implementado, tanto mediante el uso del depurador que proporciona Developer Suite, como mediante la verificación de las respuestas de la tarjeta a las diferentes excepciones y posibles entradas erróneas de los comandos.
- A continuación se listan las excepciones utilizadas en la aplicación para informar al usuario de la mala introducción de un comando:
    - Código de CLA no soportado, **6E00.**
    - Código de INS no válido, **6D00.**
    - Longitud de Lc errónea, **6A80.**

- Campos P1-P2 incorrectos, **6A86.**
- Fallo de memoria, **6581.**
- Fallo en el espacio de memoria, **6581.**
- Comando incompatible, **6981.**
- Condición incompatible, **6982.**
- Autentificación bloqueada, **6983.**
- Fichero no encontrado, **6A82**.
- Condiciones de uso no satisfechas, **6985.**
- Registro no encontrado, **6A83.**
- Verificación fallida, quedan X intentos, **63CX.**
- Finalización correcta, XX bytes por enviar, **61XX.**
- Finalización correcta, **9000.**

- Cabe señalar que la aplicación consta de un único DF (la aplicación misma), y que no se ha implementado una estructura de ficheros con capacidad de gestión de varios directorios. Al seleccionar un DF mediante el comando *Select File,* se selecciona la aplicación dentro de la tarjeta.
- Para almacenar el código secreto del directorio, se utiliza un *array* que funciona como un atributo de clase, de la misma forma que el *array read* o el *array de direccionamiento*. Esto significa que no se guarda en un fichero convencional del *array de memoria*.
- En los comandos que implementan mensajes securizados se requiere siempre el firmado y el cifrado conjunto de los datos; esto hace que la seguridad, al utilizar estos comandos, sea mayor, pero que no se puedan emplear el cifrado y el firmado por separado.

8 Conclusiones

Este trabajo demuestra la posibilidad de implementar en tecnología Java Card una aplicación que emule el comportamiento de la tarjeta de monedero electrónico WG10, pudiendo por tanto realizar las mismas funciones de una tarjeta WG10 con los requisitos de seguridad que establece la normativa aplicable a este tipo de tarjetas.

De esta manera, el resultado es una aplicación capaz de instalarse en cualquier tarjeta inteligente Java Card y que permite que dicha tarjeta pueda comportarse como monedero electrónico o, en su defecto, pueda utilizarse como tarjeta identificativa o de almacenamiento, capaz de gestionar el acceso a los ficheros contenidos en ella, calculando claves, verificando códigos y utilizando mecanismos criptográficos para el cifrado y firmado de la información en sus comunicaciones con el terminal.

Ya se han explicado a lo largo de este trabajo las múltiples ventajas que ofrece Java Card en la tecnología de las tarjetas inteligentes, permitiendo la interoperabilidad de las distintas aplicaciones gracias a implementar un sistema operativo único e independiente del fabricante. Esto provoca que se abra el mercado de las tarjetas inteligentes, y que una aplicación desarrollada en Java Card tenga potencialmente un mayor número de implantaciones en todo el mundo.

Bien es cierto que, en el sector de la seguridad de la información, el acceso a nuevos actores está limitado por razones obvias: cuanta más gente conozca las características de un estándar de pago electrónico, más probable será que se encuentren vulnerabilidades y se "piratee" el sistema. Esta es la razón por la que se guarda la confidencialidad de ciertas partes de este trabajo.

Sin embargo a día de hoy se está produciendo una revolución en las tecnologías de pago y de identificación, por lo que aparecen los pagos con el teléfono móvil mediante tecnología NFC, o la identificación por radio-frecuencia (RFID), por ejemplo. Ello hace que el sector de las tarjetas inteligentes se enfrente al reto de mantenerse como líder en el pago y la identificación electrónicos.

Gracias al uso de la tecnología Java Card se pueden abaratar aún más los costes de las tarjetas inteligentes, lo que aumenta la competitividad de esta tecnología frente a la aparición de nuevos actores en el sector.

Con la masiva informatización de los sistemas de identificación y de seguridad que se está produciendo actualmente, aparecen nuevas aplicaciones en las que utilizar la tecnología Java Card, con lo que nacen nuevas posibilidades, como puede ser el uso de esta tarjeta como identificación para el transporte público, como tarjeta sanitaria o como identificación en una empresa.

Las características de esta tarjeta permiten que se realicen transacciones financieras seguras de cantidades pequeñas, de manera que una posible aplicación podría ser su uso para el pago de las dietas de los empleados de una empresa, almacenando una cantidad fija de dinero a principio de mes y reduciendo esa cantidad a medida que se realizan los pagos.

La seguridad que aporta la norma CEN WG10 se podría exportar a otro tipo de aplicaciones, de forma que, usando las claves y características criptográficas que se han descrito en este

proyecto, se consiga hacer que aumente la seguridad de cualquier tipo de aplicación que manejaría de una forma determinada el sistema de ficheros de la tarjeta.

Realmente el uso que se le da al sistema de ficheros no se desarrolla en este proyecto, y una de las posibles evoluciones de este trabajo podría ser la implementación de un software que gestione el sistema de archivos de la tarjeta de una manera determinada y que, gracias a esto, nazca una nueva posible aplicación de la tarjeta. Este software se comunicaría con la tarjeta según las normas de ésta y mostraría al usuario opciones sencillas para gestionar el sistema de ficheros sin necesidad de utilizar lenguaje ensamblador.

Otra posible evolución en un futuro de este trabajo, sería la instalación de la aplicación desarrollada en todo tipo de dispositivos que soporten los estándares de las tarjetas inteligentes, de manera que se generalice su uso en, por ejemplo, tarjetas sin contactos, o tarjetas SIM de terminales móviles con capacidad NFC. Gracias a la tecnología Java Card, no sería problema que la aplicación conviviera con otras en la misma tarjeta.

El desarrollo de nuevos algoritmos criptográficos, como pueden ser DES-X o Advanced Encription Standard (AES), posibilita la evolución de los mecanismos de seguridad empleados en este trabajo, hasta el uso de mecanismos criptográficos más modernos y seguros, de manera que se podría llevar a cabo una mejora sobre la aplicación desarrollada.

Todas estas posibles mejoras se aprovecharían de la multitud de posibilidades que ofrece Java Card y de su conjunto de librerías para implementar aplicaciones sobre tarjetas inteligentes. Todo ello porque la tecnología Java Card permite que en la actualidad, la tarjeta chip este sea el medio de pago e identificación electrónico más extendido, gracias a ser capaz de integrar la seguridad de normas, como la CEN WG10, con la interoperabilidad y versatilidad del lenguaje Java, tal y como se ha estudiado en este trabajo.

Bibliografía

Acedo J. C., Cerezo D., Rodríguez X. R. , Sánchez R. (1999): *La Tecnología de las Tarjetas Inteligentes.* s.l. : Sánchez Reíllo R.

Chen, Z. (2000): *Java Card Technology for Smart Cards: Architecture and Programmer´s Guide.* s.l. : Pearson Education.

Dreifus H., Monk J. T. (1998) *Smart Cards: A guide to building and managing smart card applications.* s.l. : Wiley.

FNMT Dpto. Tarjetas. *Manual de uso: Tarjeta FNMT WG10.* s.l. : Fábrica Nacional de Moneda y Timbre. Versión 2.0.

Global Platform. Global Platform. [En línea] [Consultado el: 8 de agosto de 2014.] http://globalplatform.org/.

Guthery S. B., Jurgensen T. M. *Smart Card Developer´s Kit.* s.l. : Macmillan Technical Publishing.

Hassler V., Manninger M. (2002): *Java Card for E-Payment Applications*. s.l. : Artech House.

International Organization for Standarization. ISO. [En línea] [Consultado el: 25 de marzo de 2014.] http://www.iso.org/.

Oracle. Oracle. [En línea] [Consultado el: 8 de agosto de 2014.] http://www.oracle.com/java/javacard/.

Rankl W., Effing W. (2006): *Smart Card Handbook.* s.l. : Wiley.

Anexo A: Planificación y Presupuesto

A.1 Planificación

En este apartado se distribuyen las tareas necesarias para la consecución del proyecto en diferentes fases.

1) **Documentación y estudio previo**
    a. Estudio del funcionamiento de las tarjetas inteligentes (20 horas)
    b. Documentación sobre el estándar ISO 7816 (25 horas)
    c. Preparación de las herramientas de trabajo (5 horas)
2) **Aprendizaje del funcionamiento de la tarjeta WG10**
    a. Estudio del manual de uso y los comandos de la tarjeta (5 horas)
    b. Realización de prácticas utilizando el sistema de archivos de la tarjeta (10 horas)
    c. Realización de prácticas con los mecanismos de seguridad de la tarjeta (10 horas)
3) **Aprendizaje de la tecnología Java Card y el entorno de desarrollo**
    a. Estudio del lenguaje y las características de Java Card (5 horas)
    b. Desarrollo de aplicaciones sencillas (20 horas)
    c. Instalación de aplicaciones sencillas en la tarjeta Java Card (5 horas)
4) **Desarrollo de la aplicación**
    a. Interpretación de los comandos introducidos (10 horas)
    b. Implementación del sistema de ficheros (40 horas)

c. Implementación de los mecanismos de seguridad (30 horas)
d. Interconexión de las diferentes funcionalidades (20 horas)

5) Depuración del código y comprobación de excepciones

a. Comprobación de las funcionalidades en simulador (5 horas)
b. Comprobación de las respuestas de la aplicación en simulador (5 horas)

6) Instalación del *applet* y pruebas de funcionamiento

a. Instalación del *applet* en la tarjeta (1 hora)
b. Comprobación de las funcionalidades en la tarjeta (4 horas)
c. Comprobación de las respuestas de la aplicación en la tarjeta (5 horas)

7) Documentación y elaboración de la memoria

a. Redacción de la memoria (65 horas)
b. Corrección y maquetación (15 horas)

FASES	HORAS EMPLEADAS
Documentación y estudio previo	50
Aprendizaje del funcionamiento de la tarjeta WG10	25
Aprendizaje de la tecnología Java Card y del entorno de desarrollo	30
Implementación de la aplicación	100
Depuración del código y comprobación de excepciones	10
Instalación del *applet* en la tarjeta y pruebas de funcionamiento	10
Documentación y elaboración de la memoria	80
TOTAL	**305**

Tabla 2 – Desglose de tareas

A.2 Presupuesto del Trabajo Fin de Grado

A.2.1 Descripción del Trabajo

1. **Autor:** Jaime Martínez Puigvert
2. **Departamento:** Departamento de Tecnología Electrónica (Universidad Carlos III de Madrid).
3. **Descripción:** Se desarrollará un *applet* en tecnología Java Card que emule el comportamiento de la tarjeta inteligente WG10 con sistema operativo basado en ficheros y con mecanismos de seguridad.
4. **Título:** Emulación de Tarjetas Inteligentes de Monedero Electrónico Mediante Tecnología Java Card.

A.2.2 Costes de equipos y software

CONCEPTO	PRECIO DE ADQUISICIÓN (€)	TIEMPO DE USO (meses)	VIDA ÚTIL (meses)	COSTE IMPUTABLE(€)
PC con sistema operativo Windows	650	20	48	270,83
Lector de tarjetas SCM SDI010	70	15	60	17,5
Gemalto Developer Suite	500	13	60	108,33
Tarjeta Gemalto G209	10	13	No aplica	5
TOTAL				**396,66**

Tabla 3 – Costes Materiales

Fórmula para el cálculo de la amortización de los equipos y el software utilizados:

$$\mathbf{CI = PA} \times \mathbf{(TU/VU)}$$

CI= Coste Imputable (€)

TU= Tiempo de Uso (horas)

PA= Precio de Adquisición (€)

VU= Vida Útil (horas)

A.2.3 Costes de personal

OCUPACIÓN	HORAS	PRECIO/HORA (€)	IMPORTE (€)
Jefe de proyecto	25	90	2250
Ingeniero	280	55	15400
TOTAL	**305**		**17650**

Tabla 4 – Costes de Personal

A.2.4 Costes totales

CONCEPTO	PRECIO (€)
Costes materiales	396,66
Costes de personal	17650
Costes indirectos (20%)	3609,33
Subtotal	21655,99
I.V.A. (21%)	4547,75
TOTAL	**26203,74**

Tabla 5 – Costes Totales

El coste total de este proyecto es de VEINTISÉIS MIL DOSCIENTOS TRES EUROS CON SETENTA Y CUATRO CÉNTIMOS.

Leganés, 7 de Octubre de 2014

El ingeniero

Fdo. Jaime Martínez Puigvert